ENGINNERING FACULTY - COMPUTER ENGINEERING DEPARTMENT

MACHINE LEARNING
2022-2023 SPRING
FINAL PROJECT REPORT

INSTRUCTOR	: Assoc. Prof. Dr. Ahmet Çağdaş SEÇKİN
STUDENT NAME	: Elize Hamitoğlu
STUDENT ID	: 181805052
OTHER GROUP MEMBERS	: Nagihan Baz - 171805024

1 INSTRUCTIONS AND TABLE OF CONTENTS

Two main tasks will be given to the students to provide a common and unique/original dataset. The common dataset has a unique structure that is not available in known databases. The purpose of this dataset is to enable students to compete, and a ranking will be made based on the performance metric with a scoring ranging from a minimum of 5 points to a maximum of 20 points. In the unique/original dataset problem, the aim is to perform regression, classification, and clustering problems using a dataset obtained in a unique or original way. One or more datasets can be selected. Using a ready to use dataset results in a 5 points penalty. Machine learning final project instructions:

1- The project report will be prepared using the attached format. Do not forget to update the table of contents. Please do not deform the template file.
2- The report cannot exceed 20 pages (exclude cover page). Write short and concise descriptions. Points will be deducted for any content that is difficult to understand.
3- The report submission date will not be extended. Use only remote platform to submit project reports. Project report will not be delivered by mail or e-mail. No need for a printed copy.
4- Each student must submit a project report. Submitting the project report by a member of your group will mean that only that member gets points.
5- Coding will be done using python language. The submission of project codes will be accepted as a single submission for each group. Compress the project codes and your data set together as a zip and share them on the drive. Your codes should be ready to run when downloaded. During the evaluation, only your code will be run, and no settings/adjustment will be made.
6- If you have completed all the conditions above, your project will be evaluated according to the following criteria:
    a- Online exam :20 points
    b- Project and code format :15 points
    c- Common dataset project :[5, 20] points
    d- Regression application :15 points
    e- Classification application :15 points
    f- Clustering application :15 points

1.1 TABLE OF CONTENTS

2 COMMON DATASET PROJECT

2.1 DATASET AND PREPROCESSING

- **Data Source:** indoor_data_HAACS.xlsx file.
- **Data Description:** This dataset has 315 columns. X, Y and Floor columns are the outputs of this dataset. The rest of the columns are the input features.
- **Data Split:** We set train to 80% and test to 20%, set up the model and print the Mean Square Error, Variance or r-square values, test and train results.

Splitting the dataset into the Training set and Test set

In [23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=37)
```

In [108…
```
# splitting data into training and test data at 80% and 20% respectively
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
xm_train, xm_test, ym_train, ym_test = train_test_split(X, y, train_size = 0.8, random_state = 100)
```

Seperating Predictors and Response

In [18]:
```
X = comdata.drop(['X (Numeric)', 'Y (Numeric)', 'Floor (Categoric)'], axis=1)
y=comdata[['X (Numeric)', 'Y (Numeric)' , 'Floor (Categoric)']]
X.head()
```

Out[18]:

	F_dB_min	F_dB_max	F_dB_mean	F_dB_std	F_dB_MAC_ID_min	F_dB_MAC_ID_max	F_dB_CH_min	F_dB_CH_max	F_nofCH	F_nofMAC_ID	...	dB_96	MAC_ID_97	CH_97	dB_97	MAC_ID_98	CH_98	dB
0	-40	-82	-69.157895	12.024342	7	11	1	6	3	19	...	0	0	0	0	0	0	
1	-40	-82	-69.157895	12.024342	7	11	1	6	3	19	...	0	0	0	0	0	0	
2	-40	-82	-69.157895	12.024342	7	11	1	6	3	19	...	0	0	0	0	0	0	
3	-40	-82	-69.157895	12.024342	7	11	1	6	3	19	...	0	0	0	0	0	0	
4	-40	-82	-71.100000	9.164577	7	11	1	6	3	20	...	0	0	0	0	0	0	

5 rows × 312 columns

- **Data Exploration:** The exploratory data analysis performed on the dataset, including any visualizations or statistical summaries used to understand the data.

Get Data

In [80]:
```
file_loc = r"C:\Users\USER\Desktop\indoor_data_HAACS.xlsx"
comdata = pd.read_excel(file_loc)
comdata.head()
```

Out[80]:

	X (Numeric)	Y (Numeric)	Floor (Categoric)	F_dB_min	F_dB_max	F_dB_mean	F_dB_std	F_dB_MAC_ID_min	F_dB_MAC_ID_max	F_dB_CH_min	...	dB_96	MAC_ID_97	CH_97	dB_97	MAC_ID_98	CH_98	dB_98
0	0.0	0.0	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	0
1	0.0	0.0	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	0
2	0.0	0.0	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	0
3	0.0	0.0	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	0
4	0.0	0.0	0	-40	-82	-71.100000	9.164577	7	11	1	...	0	0	0	0	0	0	0

5 rows × 315 columns

Check Details

In [81]:
```
comdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1985 entries, 0 to 1984
Columns: 315 entries, X (Numeric) to  dB_99
dtypes: float64(4), int64(311)
memory usage: 4.8 MB
```

Total Number of Rows and Columns

In [82]:
```
rows_col = comdata.shape
print("Total number of records in the dataset : ", rows_col[0])
print("Total number of columns in the dataset : ", rows_col[1])
```

```
Total number of records in the dataset :  1985
Total number of columns in the dataset :  315
```

Statistical Information

In [83]:
```
comdata.describe()
```

Out[83]:

	X (Numeric)	Y (Numeric)	Floor (Categoric)	F_dB_min	F_dB_max	F_dB_mean	F_dB_std	F_dB_MAC_ID_min	F_dB_MAC_ID_max	F_dB_CH_min	...	dB_96	MAC_ID_97	CH_97	dI
count	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	1985.000000	...	1985.000000	1985.000000	1985.000000	1985.00
mean	0.781864	12.070025	1.435768	-54.832242	-86.383879	-74.489539	9.311804	29.676574	25.586398	4.672040	...	-0.820151	0.012594	0.125945	-0.88
std	9.360245	12.381293	1.035669	8.818257	2.431261	3.592711	2.618694	23.134233	21.880626	3.939659	...	8.139435	0.111544	1.115442	7.91
min	-13.200000	-6.000000	0.000000	-74.000000	-98.000000	-83.111111	3.433726	0.000000	0.000000	1.000000	...	-86.000000	0.000000	0.000000	-80.00
25%	-8.400000	0.000000	1.000000	-62.000000	-88.000000	-77.294118	7.580446	10.000000	8.000000	1.000000	...	0.000000	0.000000	0.000000	0.00
50%	1.200000	9.600000	2.000000	-56.000000	-86.000000	-74.300000	9.180852	18.000000	28.000000	6.000000	...	0.000000	0.000000	0.000000	0.00
75%	9.600000	22.800000	2.000000	-48.000000	-86.000000	-72.000000	11.144978	50.000000	33.000000	6.000000	...	0.000000	0.000000	0.000000	0.00
max	18.000000	34.800000	3.000000	-28.000000	-74.000000	-62.000000	16.599484	95.000000	96.000000	11.000000	...	0.000000	1.000000	10.000000	0.00

8 rows × 315 columns

- **Data Preprocessing:** The purpose of the code is to calculate the percent distribution of each unique value in column 'X (Numeric)' of the DataFrame comdata. The resulting Series will have unique values as indexes and their corresponding percentages as values. Then, created a new DataFrame X by removing the column 'Floor (Categoric)' from comdata.

In [9]:
```
comdata['X (Numeric)'].value_counts(normalize=True)*100
```

Out[9]:
```
-12.0    10.075567
 12.0     9.823678
 13.2     6.801008
 8.4      6.045340
-13.2     6.045340
 10.8     5.793451
 0.0      5.037783
-10.8     4.785894
-7.2      4.534005
-8.4      4.534005
 2.4      4.534005
 7.2      4.030227
 1.2      3.778338
 9.6      3.022670
 4.8      2.770781
 3.6      2.770781
-1.2      2.015113
-6.0      2.015113
-9.6      2.015113
-2.4      2.015113
-3.6      1.763224
-4.8      1.763224
 6.0      1.511335
 18.0     0.503778
 16.8     0.503778
 8.6      0.251889
 7.4      0.251889
 6.2      0.251889
-10.6     0.251889
 15.6     0.251889
 14.4     0.251889
Name: X (Numeric), dtype: float64
```

Seperating Predictors and Response

In [18]:
```
X=comdata.drop('Floor (Categoric)',axis=1) #pred
y=comdata['Floor (Categoric)']
X.head()
```

Out[18]:

	X (Numeric)	Y (Numeric)	F_dB_min	F_dB_max	F_dB_mean	F_dB_std	F_dB_MAC_ID_min	F_dB_MAC_ID_max	F_dB_CH_min	F_dB_CH_max	...	dB_96	MAC
0	0.0	0.0	-40	-82	-69.157895	12.024342	7	11	1	6	...	0	
1	0.0	0.0	-40	-82	-69.157895	12.024342	7	11	1	6	...	0	
2	0.0	0.0	-40	-82	-69.157895	12.024342	7	11	1	6	...	0	
3	0.0	0.0	-40	-82	-69.157895	12.024342	7	11	1	6	...	0	
4	0.0	0.0	-40	-82	-71.100000	9.184577	7	11	1	6	...	0	

5 rows × 314 columns

Encoding categorical data

In [19]:
```
from sklearn.preprocessing import LabelEncoder
Encoder_X = LabelEncoder()
for col in X.columns:
    X[col] = Encoder_X.fit_transform(X[col])
Encoder_y=LabelEncoder()
y = Encoder_y.fit_transform(y)
```

Data Preprocessing

In [22]:
```
# Check for missing values in the 'X (Numeric)' column
missing_values_x = comdata['X (Numeric)'].isna()

# Check for missing values in the 'Y (Numeric)' column
missing_values_y = comdata['Y (Numeric)'].isna()

# Check for missing values in the 'Floor (Categoric)' column
missing_values_floor = comdata['Floor (Categoric)'].isna()

# Count the number of missing values in each column
num_missing_x = missing_values_x.sum()
num_missing_y = missing_values_y.sum()
num_missing_floor = missing_values_floor.sum()

print(f"Number of missing values in 'X (Numeric)': {num_missing_x}")
print(f"Number of missing values in 'Y (Numeric)': {num_missing_y}")
print(f"Number of missing values in 'Floor (Numeric)': {num_missing_floor}")
```

```
Number of missing values in 'X (Numeric)': 0
Number of missing values in 'Y (Numeric)': 0
Number of missing values in 'Floor (Numeric)': 0
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
# Handling missing values
comdata.loc[:, 'X (Numeric)'].fillna(comdata['X (Numeric)'].mean(), inplace=True)
comdata.loc[:, 'Y (Numeric)'].fillna(comdata['Y (Numeric)'].mean(), inplace=True)
comdata.loc[:, 'Floor (Categoric)'].fillna('Unknown', inplace=True)

# Scaling numerical features
scaler = StandardScaler()
comdata['X (Numeric)'] = scaler.fit_transform(comdata['X (Numeric)'].values.reshape(-1, 1))
comdata['Y (Numeric)'] = scaler.fit_transform(comdata['Y (Numeric)'].values.reshape(-1, 1))

# Encoding categorical feature
comdata = pd.get_dummies(comdata, columns=['Floor (Numeric)'])

# Checking the preprocessed data
display(comdata.head())
```

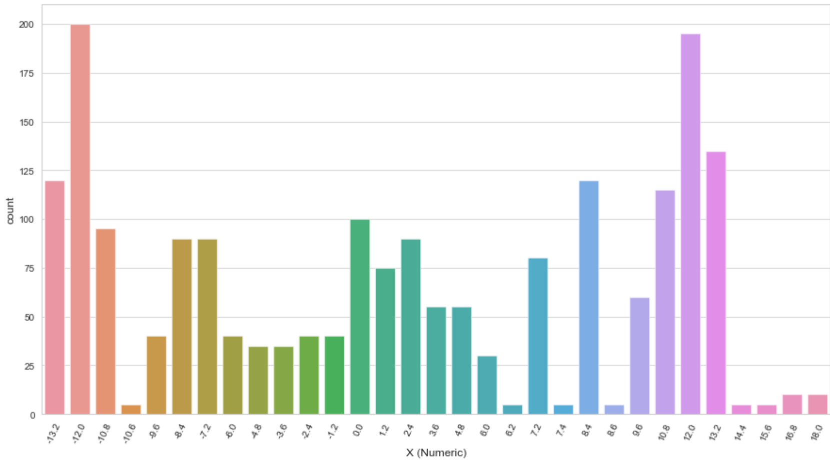
	X (Numeric)	Y (Numeric)	Floor (Categoric)	F_dB_min	F_dB_max	F_dB_mean	F_dB_std	F_dB_MAC_ID_min	F_dB_MAC_ID_max	F_dB_CH_min	...	MAC_ID_98	CH_98	dB_98	MAC_ID_99	CH_99	dB_99	(Nume
0	-0.083551	-0.975106	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	
1	-0.083551	-0.975106	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	
2	-0.083551	-0.975106	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	
3	-0.083551	-0.975106	0	-40	-82	-69.157895	12.024342	7	11	1	...	0	0	0	0	0	0	
4	-0.083551	-0.975106	0	-40	-82	-71.100000	9.164577	7	11	1	...	0	0	0	0	0	0	

5 rows × 319 columns

- **Dataset visualization:** try to explain dataset with histograms, plots, cross-correlation tables, scatter plots, etc.

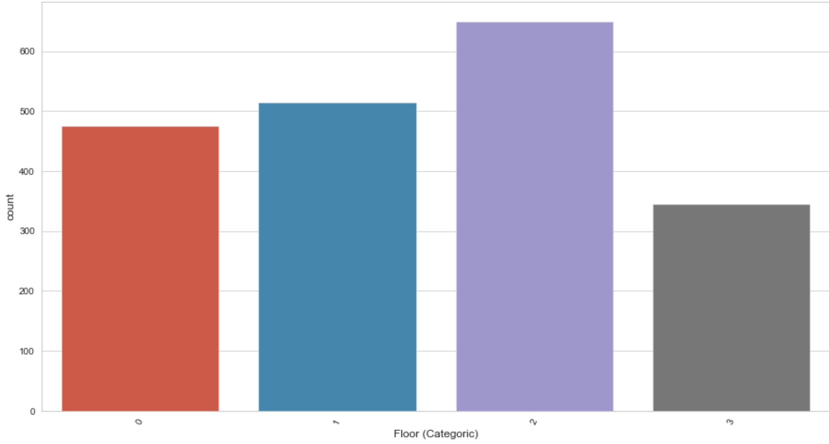
Configures the x-axis ticks, generates a countplot based on the 'X (Numeric)' column in comdata.

In [84]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (15,8))
plt.xticks(rotation=65,size=10)
sns.countplot(x='X (Numeric)', data=comdata)
plt.show()
```

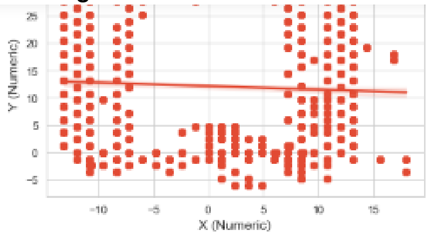


Configures the x-axis ticks, generates a countplot based on the 'Floor (Categoric)' column in comdata.

In [86]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (15,8))
plt.xticks(rotation=65,size=10)
sns.countplot(x='Floor (Categoric)', data=comdata)
plt.show()
```

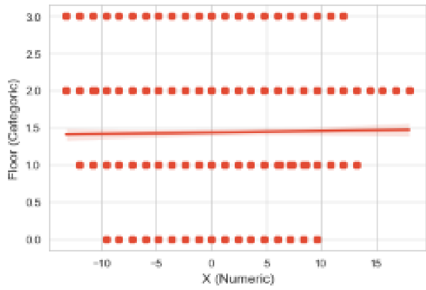


The regression plot shows a scatterplot of data points where each point represents the 'X (Numeric)' and 'Basic (Categorical)' histories. Additionally, it fits a regression line through the points to approximate the overall transition of the relationship. The slope of the regression line and the intersection point show the relationship between the two variables. For a valid regression analysis, the variable 'X (Numeric)' is expected to be numeric (continuous) and the variable 'Base (Categorical)' is expected to be categorical.



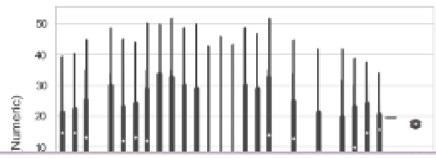
In [52]:
```
# Regression Analysis between x numeric and floor
sns.regplot('X (Numeric)', 'Floor (Categoric)', comdata)
```

Out[52]: <AxesSubplot:xlabel='X (Numeric)', ylabel='Floor (Categoric)'>



In [53]:
```
# data visualisation for
sns.violinplot(x="X (Numeric)", y="Y (Numeric)", data=comdata)
```

Out[53]: <AxesSubplot:xlabel='X (Numeric)', ylabel='Y (Numeric)'>

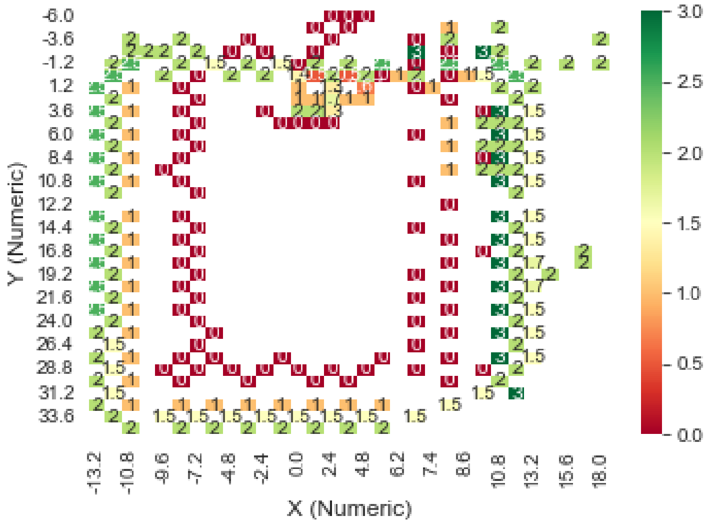


The annot=True parameter displays the actual values from the res variable, adding numeric annotations to each cell of the heatmap. This can be helpful in gaining insights from the data by directly observing the values associated with each cell.
The code creates a heatmap visualization of the res variable by applying the specified colormap and displaying the values within each cell.

Heat map

In [91]:
```
sns.heatmap(res, cmap='RdYlGn', annot=True)
```

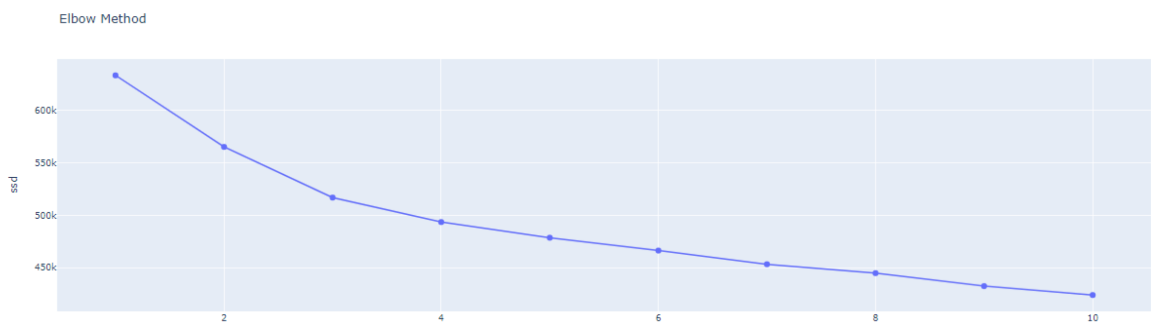
Out[91]:
```
<AxesSubplot:xlabel='X (Numeric)', ylabel='Y (Numeric)'>
```



By determining the point where the "elbow" occurs with the line graph, the optimal number of clusters for the data can be determined. The graph helps to identify the "elbow" point where the reduction in the sum of the squared distances becomes less significant as the number of clusters increases.
ssd= internal error sum of squares
k = number of clusters specified

K-Means Clustering

In [76]:
```
# create a list to store the sum of squared distances for each k
import plotly.express as px
ssd = []
scaler = StandardScaler()
full_data = scaler.fit_transform(comdata)
# fit KMeans clustering with different values of k
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(full_data)
    ssd.append(kmeans.inertia_)

# create a dataframe with the k values and corresponding ssd
comdata = pd.DataFrame({'k': range(1, 11), 'ssd': ssd})

# create the line plot using Plotly Express
fig = px.line(comdata, x='k', y='ssd', title='Elbow Method')
fig.update_traces(mode='markers+lines', marker=dict(size=8))
fig.show()
```



2.2 COMMON DATASET RESULTS AND MODEL SELECTION

With %80 Train , %20 Test	Training Result	Test Result
Logistic Regression	0,3741	0,4106
SVC	0,3892	0,7406
K-Neighbors Classifier	0,8185	0,7406
Gaussian NB	0,3728	0,4005
Decision Tree Classifier	0,9956	0,9118
Random Forest Classifier	0,9937	0,9194

With %80 Train, %20 Test	Mean Square Error	Variance or r-squared
Multiple Linear Regression	1.073825001111407	0.05417249978984362
Random Forest Regressor	0.1580468996853557	0.8604240375950214

Decision Tree Regressor	0.19164567590260287	0.8306036926949943

```
# Print Results
print("Cross-Validation Scores:", cv_scores)
print("Best Model:", best_model)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Cross-Validation Scores: [0.92138365 0.93710692 0.90251572 0.91167192 0.89905363]
Best Model: RandomForestClassifier(n_estimators=200)
Accuracy: 0.9219143576826196
Precision: 0.9152133110297411
Recall: 0.9175009666372074
F1 Score: 0.9146240858225394
```

In [74]:

```
from sklearn.metrics import r2_score

# Calculate R-squared values for X and Y regression models
r2_x_regression = r2_score(y_test, y_pred)
r2_y_regression = r2_score(y_test, y_pred)

print("r2_x_regression    r2_y_regression    accuracy")
print(r2_x_regression,r2_y_regression,accuracy)
print("-"*55)

# Calculate the performance score
project_performance_score = r2_x_regression * r2_y_regression * accuracy

print("Project Performance Score:", project_performance_score)
```

```
r2_x_regression    r2_y_regression    accuracy
0.7510502027347887 0.7510502027347887 0.9219143576826196
-------------------------------------------------------
Project Performance Score: 0.5200301384691083
```

3 ORIGINAL DATASET REGRESSION PROJECT

3.1 DATASET AND PREPROCESSING

- **Data Source:** Dataset link: https://www.kaggle.com/datasets/yasserh/walmart-dataset?select=Walmart.csv
- **Data Description:** This is the historical data that covers sales from 2010-02-05 to 2012-11-01, in the file Walmart_Store_sales. Within this file you will find the following fields:
  Store - the store number
  Date - the week of sales
  Weekly_Sales - sales for the given store
  Holiday_Flag - whether the week is a special holiday week 1 – Holiday week 0 – Non-holiday week
  Temperature - Temperature on the day of sale
  Fuel_Price - Cost of fuel in the region
  CPI – Prevailing consumer price index
  Unemployment - Prevailing unemployment rate
  Holiday Events\
  Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13\
  Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13\
  Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13\
  Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13
- **ML Problem definition:** The regression problem is to estimate the weekly sales of Walmart stores based on the available features in the dataset. The target variable for the regression problem is 'Weekly_Sales'. The remaining columns in the dataset, such as 'Store', 'Date', 'Temperature', 'Fuel_Price', 'CPI', and 'Unemployment', can be considered as potential features.
- **Data Split:** We set the train to 80% and the test to 20%, build the model and print the Mean Square Error and Variance or r-squared values.

In [38]:

```
# splitting data into training and test data at 80% and 20% respectively
from sklearn.model_selection import train_test_split
xm_train, xm_test, ym_train, ym_test = train_test_split(X, y, train_size = 0.8, random_state = 100)
```

- **Data Exploration:** The exploratory data analysis performed on the dataset, including any visualizations or statistical summaries used to understand the data.

```
file_loc = r"C:\Users\USER\Desktop\Walmart (2).xlsx"
df = pd.read_excel(file_loc)
```

In [7]:

```
df.head()
```

Out[7]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	2010-02-05	1643690.90	0	42.31	2572	2.110964e+09	8106
1	1	2010-02-12	1641957.44	1	38.51	2548	2.112422e+09	8106
2	1	2010-02-19	1611968.17	0	39.93	2514	2.112891e+09	8106
3	1	2010-02-26	1409727.59	0	46.63	2561	2.113196e+09	8106
4	1	2010-03-05	1554806.68	0	46.5	2625	2.113501e+09	8106

In [8]:

```
# Finding information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   object
 5   Fuel_Price    6435 non-null   object
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(2), object(3)
memory usage: 402.3+ KB
```

- **Data Preprocessing:** The preprocessing steps applied to the dataset, including any missing value imputation, feature scaling, or feature selection techniques.
  This code separates the target variable 'Weekly_Sales' from the DataFrame df and assigns it to the variable y. It also creates a new DataFrame x that excludes the 'Weekly_Sales' column, which contains the predictors or independent variables. This separation is typically done to prepare the data for training a machine learning model or performing statistical analysis, where the target variable (y) and predictors (x) are treated separately.

[36]:

```
# Separating target variable and predictors
y = df ['Weekly_Sales']
x = df.drop(['Weekly_Sales'], axis =1)
```

This code snippet converts the 'Date' column in a DataFrame from a non-datetime format to a datetime format using pandas' to_datetime() function. This allows for easier manipulation and analysis of dates and times within the DataFrame. Then, reframes the columns in the DataFrame df by extracting the weekday, month, and year components from the 'Date' column. It then drops the 'Date' column and assigns the target variable name to target. The remaining column names, excluding the target variable, are stored in features. Finally, a deep copy of df is created as original_df.

We start our data cleaning from here

In [9]:

```
# converting date object to datetime
df['Date'] = pd.to_datetime(df.Date)
df.head()
```

Out[9]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	2010-02-05	1643690.90	0	42.31	2572	2.110964e+09	8106
1	1	2010-02-12	1641957.44	1	38.51	2548	2.112422e+09	8106
2	1	2010-02-19	1611968.17	0	39.93	2514	2.112891e+09	8106
3	1	2010-02-26	1409727.59	0	46.63	2561	2.113196e+09	8106
4	1	2010-03-05	1554806.68	0	46.5	2625	2.113501e+09	8106

In [10]:

```
# Reframing the columns by breaking the date into weeks, month and year for analysis

df['weekday'] = df.Date.dt.weekday
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year

df.drop(['Date'], axis=1, inplace=True)#,'month'

target = 'Weekly_Sales'
features = [i for i in df.columns if i not in [target]]
original_df = df.copy(deep=True)

df.head()
```

Out[10]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	weekday	month	year
0	1	1643690.90	0	42.31	2572	2.110964e+09	8106	4	2	2010

Feature engineering: Feature engineering is the process of selecting and transforming variables (features) in a dataset to improve the performance of a machine learning model. The goal of feature engineering is to create features that are relevant, informative, and non-redundant, and that capture the key relationships between variables in the dataset.
The code reframes the columns in the DataFrame df by extracting the weekday, month, and year components from the 'Date' column. It then drops the 'Date' column and assigns the target variable name to target. The remaining column names, excluding the target variable, are stored in features. Finally, a deep copy of df is created as original_df.

In [10]:
```python
# Reframing the columns by breaking the date into weeks, month and year for analysis

df['weekday'] = df.Date.dt.weekday
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year

df.drop(['Date'], axis=1, inplace=True)#, 'month'

target = 'Weekly_Sales'
features = [i for i in df.columns if i not in [target]]
original_df = df.copy(deep=True)

df.head()
```

Out[10]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	weekday	month	year
0	1	1643690.90	0	42.31	2572	2.110964e+09	8106	4	2	2010
1	1	1641957.44	1	38.51	2548	2.112422e+09	8106	4	2	2010
2	1	1611968.17	0	39.93	2514	2.112891e+09	8106	4	2	2010
3	1	1409727.59	0	46.63	2561	2.113196e+09	8106	4	2	2010
4	1	1554806.68	0	46.5	2625	2.113501e+09	8106	4	3	2010

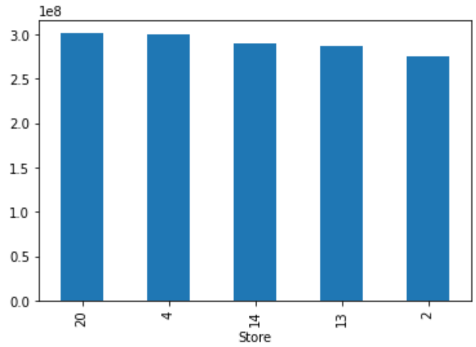
- **Dataset visualization:** try to explain dataset with histograms, plots, cross-correlation tables, scatter plots, etc. **This code computes the sum of weekly sales for each store, sorts the sums in descending order, and returns the top 5 stores with the highest weekly sales.**

In [15]:
```python
# The stores with the highest weekly sales
df.groupby(['Store'])['Weekly_Sales'].sum().sort_values(ascending=False).head(5)
```

Out[15]:
```
Store
20    3.013978e+08
4     2.995440e+08
14    2.889999e+08
13    2.865177e+08
2     2.753824e+08
Name: Weekly_Sales, dtype: float64
```

In [16]:
```python
# The stores with the highest weekly sales visualization
df.groupby(['Store'])['Weekly_Sales'].sum().sort_values(ascending=False).head(5).plot(kind='bar')
```

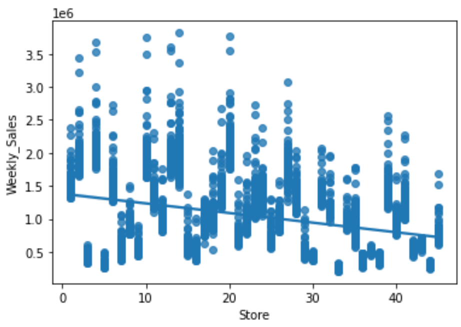
Out[16]: <AxesSubplot:xlabel='Store'>



Using sns.regplot() we visualized the relationship between the 'Store' variable and the 'Weekly_Sales' variable in the DataFrame df using a regression row. The regression line can help you understand the nature of the relationship and identify trends or patterns between two variables.

```
# Regression Analysis between sales and store
sns.regplot('Store', 'Weekly_Sales', df)
```

Out[28]: <AxesSubplot:xlabel='Store', ylabel='Weekly_Sales'>



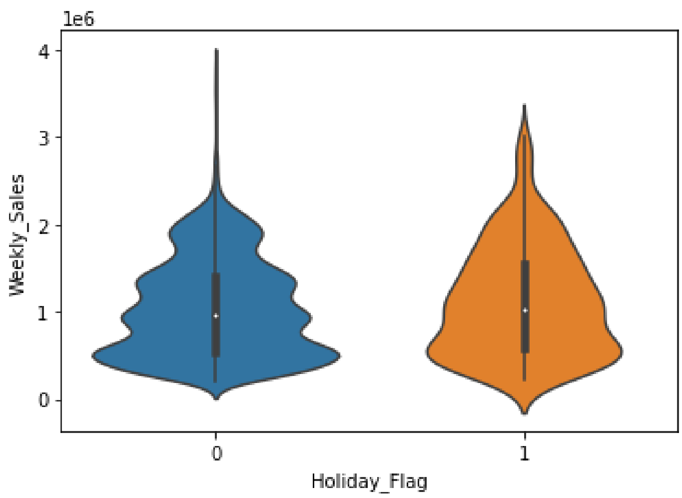
In [29]:
```
# Regression Analysis between sales and Holiday_Flag
sns.regplot('Holiday_Flag', 'Weekly_Sales', df)
```

Out[29]: <AxesSubplot:xlabel='Holiday_Flag', ylabel='Weekly_Sales'>

Using sns.violinplot() we visualized 'Weekly_Sales' distributions for different categories of 'Holiday_Flag' variable in DataFrame df. The fiddle drawing helps to understand the shape, distribution and central trend of the 'Weekly_Sales' variable in different holiday categories.

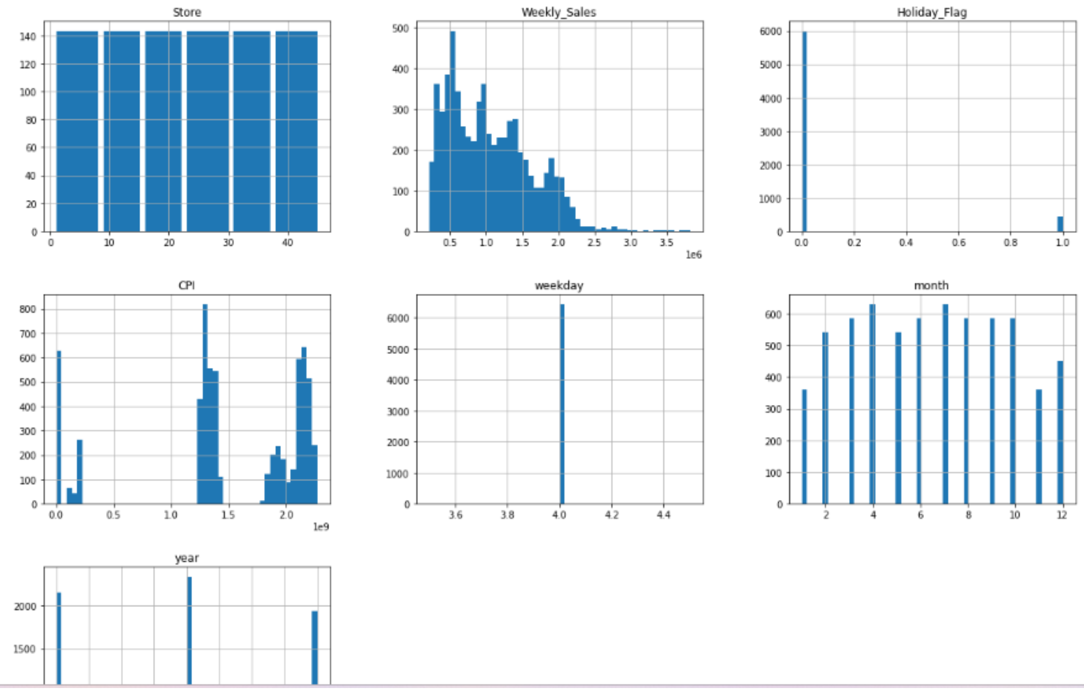
In [31]:
```
# data visualisation for
sns.violinplot(x="Holiday_Flag", y="Weekly_Sales", data=df)
```

Out[31]: <AxesSubplot:xlabel='Holiday_Flag', ylabel='Weekly_Sales'>



Using df.hist() we visualized the data distribution for each column in the DataFrame df. Each column will have its own histogram showing the frequency or number of data points falling into each bin.

```python
# Histogram for all columns
import matplotlib.pyplot as plt
df.hist(bins=50, figsize=(20,15))
plt.show()
```



3.2 REGRESSION RESULTS AND MODEL SELECTION

This code snippet applies standardization or scaling to the data in the variable x using the StandardScaler from scikit-learn. It computes the mean and standard deviation of x and then transforms the data by subtracting the mean and dividing by the standard deviation. The resulting standardized data is assigned to the variable X. Finally, X[0:1] retrieves the first row of the standardized data array for further inspection or analysis.

Model Building and Selection

In [36]:
```python
# Separating target variable and predictors
y = df ['Weekly_Sales']
x = df.drop(['Weekly_Sales'], axis =1)
```

In [37]:
```python
# Normalization data to bring all values to common scale
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(x).transform(x)
X[0:1]
```

Out[37]:
```
array([[-1.08032351, -1.6768291 ,  0.88484192,  0.11639643, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.15075567, -0.15075567,
        -0.15075567, -0.15075567, -0.15075567, -0.27420425,  3.30403793,
        -0.31622777, -0.32943456, -0.30265996, -0.31622777, -0.32943456,
        -0.31622777, -0.31622777, -0.31622777, -0.24343225, -0.27420425,
        -0.75592895, -0.65574385]])
```

In [38]:
```python
# splitting data into training and test data at 80% and 20% respectively
from sklearn.model_selection import train_test_split
xm_train, xm_test, ym_train, ym_test = train_test_split(X, y, train_size = 0.8, random_state = 100)
```

MODEL	Mean Square Error	Variance or r-squared
Multiple Linear Regression	18493107679.295235	0.9429018333238277
Random Forest Regressor	17536449249.215878	0.9457754621459217
Decision Tree Regressor	20811360827.70793	0.9356848276001741

We can say that the best model is the Random Forest by looking at the values.

3.3 REGRESSION HYPERPARAMTER OPTIMIZATION RESULTS

This code performs hyperparameter tuning for a random forest regression model using grid search. It defines a grid of hyperparameters to search over, creates an instance of the random forest regressor, sets up the GridSearchCV object with the necessary parameters, fits the object to the data, and prints the results of the grid search. The grid search helps to find the best combination of hyperparameters that optimize the performance of the model based on the provided evaluation metric.

Tuning our model before deployment

In [51]:
```
# Tuning our model
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(X, y)
print(grid_search)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [52]:
```
# obtaining the best parameters
grid_search.best_params_
```

Out[52]:
```
{'max_features': 6, 'n_estimators': 30}
```

In [53]:
```
# obtaining the best estimators
grid_search.best_estimator_
```

Out[53]:
```
RandomForestRegressor(max_features=6, n_estimators=30)
```

The loop iterates over each parameter combination and prints the corresponding MSE and hyperparameter values. By printing the MSE for each parameter combination, we can evaluate the performance of different parameter settings. This information helps in understanding the impact of different hyperparameters on the model's performance.

In [54]:
```
# printing all MSE for each parameter combinations
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
698845.5981393111 {'max_features': 2, 'n_estimators': 3}
654392.6814014538 {'max_features': 2, 'n_estimators': 10}
636187.4016068892 {'max_features': 2, 'n_estimators': 30}
676467.9014538403 {'max_features': 4, 'n_estimators': 3}
641477.6799379607 {'max_features': 4, 'n_estimators': 10}
630283.8001052317 {'max_features': 4, 'n_estimators': 30}
666438.6093668051 {'max_features': 6, 'n_estimators': 3}
626478.3079380571 {'max_features': 6, 'n_estimators': 10}
607715.8792812905 {'max_features': 6, 'n_estimators': 30}
642656.8134494207 {'max_features': 8, 'n_estimators': 3}
609862.1155464306 {'max_features': 8, 'n_estimators': 10}
608552.6452037573 {'max_features': 8, 'n_estimators': 30}
680915.1800373765 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
659532.2617214441 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
682931.2806871571 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
642155.878190613 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
680829.9120556896 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
624885.8025437042 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

4 ORIGINAL DATASET CLASSIFICATION PROJECT

4.1 DATASET AND PREPROCESSING

- **Data Source:** Dataset link: https://www.kaggle.com/datasets/ppb00x/credit-risk-customers?select=credit_customers.csv

- **Data Description:** This dataset classifies people described by a set of attributes as good or bad credit risks. Within this file you will find the following fields:
  checking_status: Status of existing checking account
  duration: Duration in months
  credit_history: credits taken, paid back duly, delays, critical accounts
  purpose: Purpose of the credit
  credit_amount: Amount of credit
  savings_status: Status of savings account/bond
  employment: Present employment, in number of years
  installment_commitment: Installment rate in percentage of disposable income
  personal_status: sex and marital data
  other_parties: Other debtors / guarantors
- **ML Problem definition:** The purpose of this task is to predict whether an applicant is likely to default on their loan based on a number of characteristics such as credit history, savings status, employment, age, and other factors.
- **Data Split:** We set the train to 80% and the test to 20%, build the model and print the Mean Squared Error and Variance or r-squared values and calculate the classification models.

-

In [17]:
```
# splitting data into training and test data at 80% and 20% respectively
from sklearn.model_selection import train_test_split
xm_train, xm_test, ym_train, ym_test = train_test_split(X, y, train_size = 0.8, random_state = 100)
```

- **Data Exploration:** The exploratory data analysis performed on the dataset, including any visualizations or statistical summaries used to understand the data.

```
file_loc = r"C:\Users\USER\Desktop\credit_customers (1).xlsx"
credit_customers_data = pd.read_excel(file_loc)
credit_customers_data.head()
```

Out[2]:

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitment	personal_status	other_parties	...	property_magnitude	age	other_p
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.0	male single	none	...	real estate	67.0	
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.0	female div/dep/mar	none	...	real estate	22.0	
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.0	male single	none	...	real estate	49.0	
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.0	male single	guarantor	...	life insurance	45.0	
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.0	male single	none	...	no known property	53.0	

5 rows × 21 columns

In [3]:

```
rows_col = credit_customers_data.shape
print("Total number of records in the dataset : ", rows_col[0])
print("Total number of columns in the dataset : ", rows_col[1])
```

```
Total number of records in the dataset :  1000
Total number of columns in the dataset :  21
```

In [4]:

```
credit_customers_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   checking_status         1000 non-null   object 
 1   duration                1000 non-null   float64
 2   credit_history          1000 non-null   object 
 3   purpose                 1000 non-null   object 
 4   credit_amount           1000 non-null   float64
 5   savings_status          1000 non-null   object 
 6   employment              1000 non-null   object 
 7   installment_commitment  1000 non-null   float64
 8   personal_status         1000 non-null   object 
 9   other_parties           1000 non-null   object 
 10  residence_since         1000 non-null   float64
 11  property_magnitude      1000 non-null   object 
 12  age                     1000 non-null   float64
 13  other_payment_plans     1000 non-null   object 
 14  housing                 1000 non-null   object 
 15  existing_credits        1000 non-null   float64
 16  job                     1000 non-null   object 
 17  num_dependents          1000 non-null   float64
 18  own_telephone           1000 non-null   object 
 19  foreign_worker          1000 non-null   object 
 20  class                   1000 non-null   object 
dtypes: float64(7), object(14)
memory usage: 164.2+ KB
```

- **Data Preprocessing:** The preprocessing steps applied to the dataset, including any missing value imputation, feature scaling, or feature selection techniques.
  Creates a new DataFrame X, dropping the 'class' column from the original Credit_customers_data DataFrame. Assigns the 'class' column of credit_customers_data to variable y. Represents the target variable or labels we want to predict. Creates an instance of the StandardScaler class that will be used to scale features. It fits the scaler to the data by calculating the mean and standard deviation of each feature. It then transforms the features by subtracting the mean and dividing by the standard deviation. The resulting variable Xa holds the scaled properties.

In [13]:

```
from sklearn.preprocessing import StandardScaler
X = credit_customers_data.drop(['class'], axis=1)
y = credit_customers_data['class']
std_scaler = StandardScaler()
Xa = std_scaler.fit_transform(X)
```

- **Feature engineering:** Feature engineering is the process of selecting and transforming variables (features) in a dataset to improve the performance of a machine learning model. The goal of feature engineering is to create features that are relevant, informative, and non-redundant, and that capture the key relationships between variables in the dataset.
  This code snippet manipulates the 'personal_status' column in the credit_customers_data DataFrame. It splits the values in the column into two separate columns, 'sex' and 'marriage'. Then, it drops the

original 'personal_status' column from the DataFrame, resulting in a modified DataFrame with the 'sex' and 'marriage' columns.

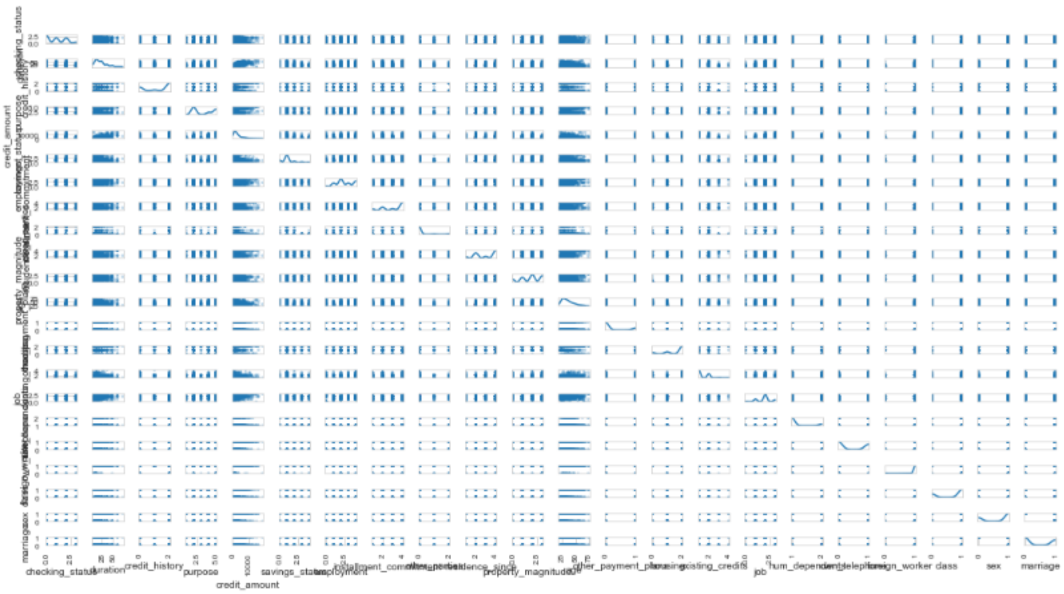
In [6]:
```
credit_customers_data[['sex', 'marriage']] = credit_customers_data.personal_status.str.split(" ", expand = True)
credit_customers_data.drop(['personal_status'], axis=1, inplace = True)
```

This code is performing replacement or mapping of categorical values with numerical values in several columns of the credit_customers_data DataFrame.

In [8]:
```
credit_customers_data['checking_status'].replace(['no checking', '<0', '0<=X<200', '>=200'], [0,1,2,3], inplace = True)
credit_customers_data['credit_history'].replace(['critical/other existing credit', 'delayed previously' , 'existing paid', 'no credits/all paid', 'all paid'], [0,1,2,2,2]
credit_customers_data['purpose'].replace(['business', 'new car','used car', 'education', 'retraining', 'other','domestic appliance','radio/tv','furniture/equipment','repa
credit_customers_data['savings_status'].replace(['no known savings', '<100','100<=X<500','500<=X<1000','>=1000'], [0,1,2,3,4], inplace = True)
credit_customers_data['employment'].replace(['unemployed', '<1','1<=X<4','4<=X<7','>=7'], [0,1,2,3,4], inplace = True)
credit_customers_data['other_parties'].replace(['none', 'co applicant', 'guarantor'], [0,1,2], inplace = True)
credit_customers_data['property_magnitude'].replace(['no known property', 'life insurance', 'car', 'real estate'], [0,1,2,3], inplace = True)
credit_customers_data['other_payment_plans'].replace(['none', 'stores', 'bank'], [0,1,1], inplace = True)
credit_customers_data['housing'].replace(['for free', 'rent', 'own'], [0,1,2], inplace = True)
credit_customers_data['job'].replace(['unemp/unskilled non res', 'unskilled resident', 'skilled', 'high qualif/self emp/mgmt'], [0,1,2,3], inplace = True)
credit_customers_data['own_telephone'].replace(['yes', 'none'], [1,0], inplace = True)
credit_customers_data['foreign_worker'].replace(['yes', 'no'], [1,0], inplace = True)
credit_customers_data['class'].replace(['good', 'bad'], [1,0], inplace = True)
credit_customers_data['sex'].replace(['male', 'female'], [1,0], inplace = True)
credit_customers_data['marriage'].replace(['single', 'div/sep','div/dep/mar','mar/wid'], [0,0,1,1], inplace = True)
```

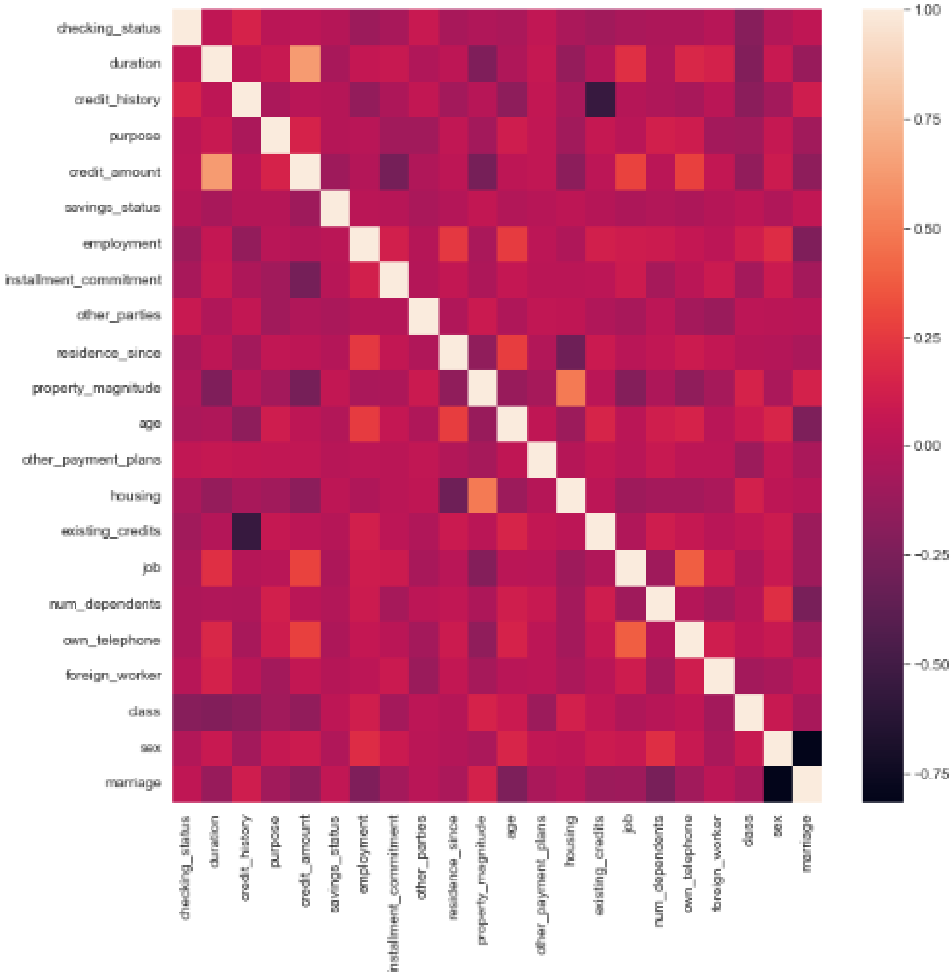
- **Dataset visualization:** try to explain dataset with histograms, plots, cross-correlation tables, scatter plots, etc. **This code snippet creates a scatter matrix plot to visualize the pairwise relationships and distributions of variables in the credit_customers_data DataFrame. The resulting plot provides a quick overview of the data and allows for visual examination of potential relationships or patterns between variables.**

In [47]:
```
pd.plotting.scatter_matrix(credit_customers_data, alpha=0.3,figsize=(15,8),diagonal='kde' )
plt.tight_layout()
```



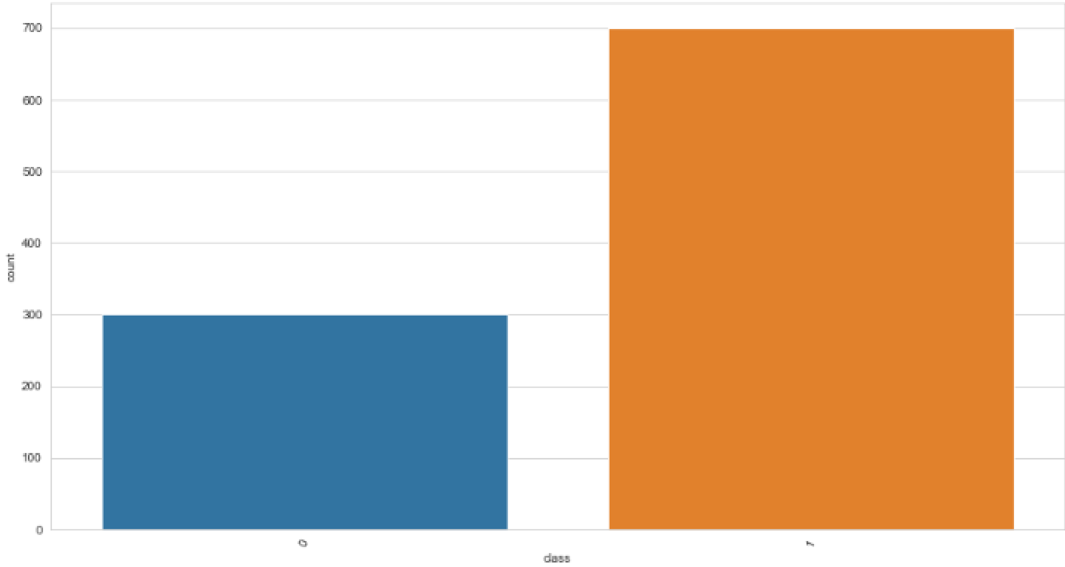
This code snippet creates a heatmap plot of the correlation matrix for the variables in the credit_customers_data DataFrame using seaborn and matplotlib.

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

sns.heatmap(credit_customers_data.corr())

# df.drop(['credit_amount','duration'],axis=1,inplace=True)

plt.show()
```



This code snippet sets the seaborn style to "whitegrid", creates a countplot of the 'class' column in the credit_customers_data DataFrame, adjusts the appearance of the plot's x-axis tick labels.

In [28]:
```
sns.set_style("whitegrid")
plt.figure(figsize = (15,8))
plt.xticks(rotation=65,size=10)
sns.countplot(x='class', data=credit_customers_data)
plt.show()
```



4.2 CLASSIFICATION RESULTS AND MODEL SELECTION

With %80 Train , %20 Test Model	Training Result	Test Result
Logistic Regression	0,7338	0,7000
SVC	0,8588	0,7550
K-Neighbors Classifier	0,8087	0,6950
Gaussian NB	0,7375	0,6800
Decision Tree Classifier	1,0	0,7300
Random Forest Classifier	1,0	0,7700

We can say that the best model is the Random Forest by looking at the values.

5 ORIGINAL DATASET CLUSTERING PROJECT

5.1 DATASET AND PREPROCESSING

- **Data Source:** Dataset link: https://www.kaggle.com/datasets/fatihb/coffee-quality-data-cqi?select=df_arabica_clean.csv
- **Data Description:** This is the cleaned, tabular data for arabica-type coffees. It consists of 41 columns.
- **ML Problem definition:** In this project, we aimed to build a predictive model to predict total cup scores, a comprehensive measure of coffee quality based on these influencing factors.
- **Data Split:** We set the train to 80% and the test to 20%.

```
# Split the data into train and test sets
X = df.drop('Total Cup Points', axis=1)
y = df['Total Cup Points']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Data Exploration:** The exploratory data analysis performed on the dataset, including any visualizations or statistical summaries used to understand the data.

```
In [2]: df = pd.read_excel(r"C:\Users\USER\Desktop\df_arabica_clean.xlsx")
```

```
In [3]: df.head()
```

Out[3]:

	Unnamed: 0	ID	Country of Origin	Farm Name	Lot Number	Mill	ICO Number	Company	Altitude	Region	...	Total Cup Points	Moisture Percentage	Category One Defects	Quakers	Color	Category Two Defects	Expiration	Certification Body	Certification Address	Certification Contact
0	0	0	Colombia	Finca El Paraiso	CQU2022015	Finca El Paraiso	NaN	Coffee Quality Union	1700-1930	Piendamo,Cauca	...	89.33	2023-08-11 00:00:00	0	0	green	3	September 21st, 2023	Japan Coffee Exchange	ã€'413-0002 é□™å²¡çœŒç†±æµ·å¸‚ä¼Šè±†å±±ï¼ï¼'ï...	æ□¾æ¾¤ã€€å®□æ¨ã€€Koju Matsuzawa - +81(0)90856...
1	1	1	Taiwan	Royal Bean Geisha Estate	The 2022 Pacific Rim Coffee Summit,T037	Royal Bean Geisha Estate	NaN	Taiwan Coffee Laboratory	1200	Chiayi	...	87.58	2023-05-10 00:00:00	0	0	blue-green	0	November 15th, 2023	Taiwan Coffee Laboratory å□°ç□£å'–å•¡ç ”ç©¶å®¤	QAHWAH CO., LTD 4F, No. 225, Sec. 3, Beixin Rd...	Lin, Jen-An Neil æ□—ä»□å®‰ - 886-289116612
2	2	2	Laos	OKLAO coffee farms	The 2022 Pacific Rim Coffee Summit,LA01	oklao coffee processing plant	NaN	Taiwan Coffee Laboratory	1300	Laos Borofen Plateau	...	87.42	2023-04-10 00:00:00	0	0	yellowish	2	November 15th, 2023	Taiwan Coffee Laboratory å□°ç□£å'–å•¡ç ”ç©¶å®¤	QAHWAH CO., LTD 4F, No. 225, Sec. 3, Beixin Rd...	Lin, Jen-An Neil æ□—ä»□å®‰ - 886-289116612
3	3	3	Costa Rica	La Cumbre	CQU2022017	La Montana Tarrazu Mill	NaN	Coffee Quality Union	1900	Los Santos,Tarrazu	...	87.17	2023-08-11 00:00:00	0	0	green	0	September 21st, 2023	Japan Coffee Exchange	ã€'413-0002 é□™å²¡çœŒç†±æµ·å¸‚ä¼Šè±†å±±ï¼ï¼'ï...	æ□¾æ¾¤ã€€å®□æ¨ã€€Koju Matsuzawa - +81(0)90856...
4	4	4	Colombia	Finca Santuario	CQU2023002	Finca Santuario	NaN	Coffee Quality Union	1850-2100	Popayan,Cauca	...	87.08	2023-06-11 00:00:00	0	2	yellow-green	2	March 5th, 2024	Japan Coffee Exchange	ã€'413-0002 é□™å²¡çœŒç†±æµ·å¸‚ä¼Šè±†å±±ï¼ï¼'ï...	æ□¾æ¾¤ã€€å®□æ¨ã€€Koju Matsuzawa - +81(0)90856...

5 rows × 41 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 207 entries, 0 to 206
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             207 non-null    int64
 1   ID                     207 non-null    int64
 2   Country of Origin      207 non-null    object
 3   Farm Name              205 non-null    object
 4   Lot Number             206 non-null    object
 5   Mill                   204 non-null    object
 6   ICO Number             75 non-null     object
 7   Company                207 non-null    object
 8   Altitude               206 non-null    object
 9   Region                 205 non-null    object
 10  Producer               206 non-null    object
 11  Number of Bags         207 non-null    int64
 12  Bag Weight             207 non-null    object
 13  In-Country Partner     207 non-null    object
 14  Harvest Year           207 non-null    object
 15  Grading Date           207 non-null    object
 16  Owner                  207 non-null    object
 17  Variety                201 non-null    object
 18  Status                 207 non-null    object
 19  Processing Method      202 non-null    object
 20  Aroma                  207 non-null    object
 21  Flavor                 207 non-null    object
 22  Aftertaste             207 non-null    object
 23  Acidity                207 non-null    object
 24  Body                   207 non-null    object
 25  Balance                207 non-null    object
 26  Uniformity             207 non-null    object
 27  Clean Cup              207 non-null    float64
 28  Sweetness              207 non-null    float64
 29  Overall                207 non-null    object
 30  Defects                207 non-null    float64
 31  Total Cup Points       207 non-null    float64
 32  Moisture Percentage    207 non-null    object
 33  Category One Defects   207 non-null    int64
 34  Quakers                207 non-null    int64
 35  Color                  207 non-null    object
 36  Category Two Defects   207 non-null    int64
 37  Expiration             207 non-null    object
 38  Certification Body     207 non-null    object
 39  Certification Address  207 non-null    object
 40  Certification Contact  207 non-null    object
dtypes: float64(4), int64(6), object(31)
memory usage: 66.4+ KB
```

- **Data Preprocessing:** The preprocessing steps applied to the dataset, including any missing value imputation, feature scaling, or feature selection techniques.
  This code snippet identifies and filters the duplicated rows in the DataFrame df, creates a new DataFrame duplicate_rows_data containing only the duplicated rows, and then prints the count of duplicate rows. This can be useful for identifying and handling duplicate data in a dataset.

```
In [5]: # Handle duplicates
duplicate_rows_data = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_data.shape)
```

```
number of duplicate rows:  (0, 41)
```

This code calculates the missing ratio for each column in the DataFrame df, identifies the columns with non-zero missing ratios, sorts them in descending order, selects the top 30 columns with the highest missing ratios, and displays them in a DataFrame called missing_data. This allows for easy inspection

of columns with significant amounts of missing data.

In [7]:
```
#check missing ratio
data_na = (df.isnull().sum() / len(df)) * 100
data_na = data_na.drop(data_na[data_na == 0].index).sort_values(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :data_na})
missing_data.head(20)
```

Out[7]:

	Missing Ratio
ICO Number	63.768116
Variety	2.898551
Processing Method	2.415459
Mill	1.449275
Farm Name	0.966184
Region	0.966184
Lot Number	0.483092
Altitude	0.483092
Producer	0.483092

1) This code snippet manually imputes specific values for the 'Altitude' column based on the 'ID' column. It then defines a function to clean and calculate the mean for each value in the 'Altitude' column, and applies that function to clean and update the 'Altitude' column with the cleaned and calculated mean values.
2) This code extracts the prior year from the 'Harvest Year' column by splitting each value on the '/' delimiter, accessing the first element (which represents the prior year), and removing any leading or trailing whitespace. The 'Harvest Year' column is then updated with the extracted prior year values.
3) These lines of code convert the 'Harvest Year' column to datetime objects using the specified format ('%Y') and convert the 'Expiration' column to datetime objects using the parser.parse function. This ensures that the 'Harvest Year' and 'Expiration' columns are represented as datetime objects, allowing for easier manipulation and analysis of date and time data.
4) This code calculates the difference in days between the 'Expiration' and 'Harvest Year' columns and stores the result in a new column called 'Coffee Age'. The 'Coffee Age' column represents the age of the coffee in days, indicating the time elapsed between the harvest year and the expiration date.
5) This code snippet drops the specified columns from the DataFrame df using the drop() function, effectively removing those columns from the dataset. This can be useful when certain columns are not relevant or necessary for the analysis or when they contain redundant information.

In [9]:
```
# Manually impute specific values based on ID (Which we cant use function)
df.loc[df['ID'] == 99, 'Altitude'] = 5273  # Impute value for ID 99
df.loc[df['ID'] == 105, 'Altitude'] = 1800  # Impute value for ID 105
df.loc[df['ID'] == 180, 'Altitude'] = 1400  # Impute value for ID 180


# Define a function to clean and calculate the mean
def clean_altitude_range(range_value):
    if isinstance(range_value, str):
        range_value = range_value.replace(" ", "")  # Remove blank spaces
        if '-' in range_value:
            try:
                start, end = range_value.split('-')
                start = int(start)
                end = int(end)
                return (start + end) / 2
            except ValueError:
                return np.nan
        else:
            try:
                return int(range_value)
            except ValueError:
                return np.nan
    else:
        return range_value

# Apply the function to clean and calculate the mean for each value in the "Altitude" column
df['Altitude'] = df['Altitude'].apply(clean_altitude_range)
```

In [10]:
```
# Extract the prior year from the "Harvest Year" column
df['Harvest Year'] = df['Harvest Year'].str.split('/').str[0].str.strip()
```

In [11]:
```
# Convert "Harvest Year" and "Expiration" columns to datetime objects using dateutil parser
df['Harvest Year'] = pd.to_datetime(df['Harvest Year'], format='%Y')
df['Expiration'] = df['Expiration'].apply(parser.parse)
```

In [12]:
```
# Calculate the difference in days between "Expiration" and "Harvest Year" columns
df['Coffee Age'] = (df['Expiration'] - df['Harvest Year']).dt.days
```

In [13]:
```
columns_to_drop = ['ID','ICO Number','Owner','Region','Certification Contact','Certification Address','Farm Name',"Lot Number","Mill","ICO Number","Producer",'Company','Expiration', 'Harvest Year',
                   "Unnamed: 0",'Number of Bags','Bag Weight','In-Country Partner','Grading Date','Variety','Status','Defects','Uniformity','Clean Cup','Sweetness','Certification Body']
df.drop(columns_to_drop, axis=1, inplace=True)
```

- **Dataset visualization:** try to explain dataset with histograms, plots, cross-correlation tables, scatter plots, etc.
  This code generates a grid of histograms for the numeric attributes specified in the numeric_attributes list. Each histogram represents the distribution of values for a specific numeric attribute in the DataFrame df.

In [14]:

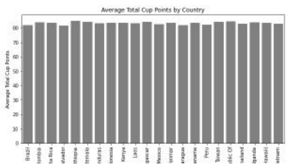
```
# List of numeric attributes
numeric_attributes = ['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Overall', 'Total Cup Points', 'Moisture Percentage','Coffee Age']

# Create a subplot for each numeric attribute
fig = make_subplots(rows=len(numeric_attributes), cols=1)

# Add a histogram to the subplot for each numeric attribute
for i, attribute in enumerate(numeric_attributes):
    fig.add_trace(go.Histogram(x=df[attribute], nbinsx=50, name=attribute), row=i+1, col=1)

fig.update_layout(height=200*len(numeric_attributes), width=800, title_text="Histograms of Numeric Attributes")
fig.show()
```

Histograms of Numeric Attributes

This code calculates the average 'Total Cup Points' for each country in the DataFrame df. It then visualizes the results using a Choropleth map and a bar plot. The Choropleth map shows the average 'Total Cup Points' for each country on a geographical map, while the bar plot provides a visual comparison of the average scores between countries.

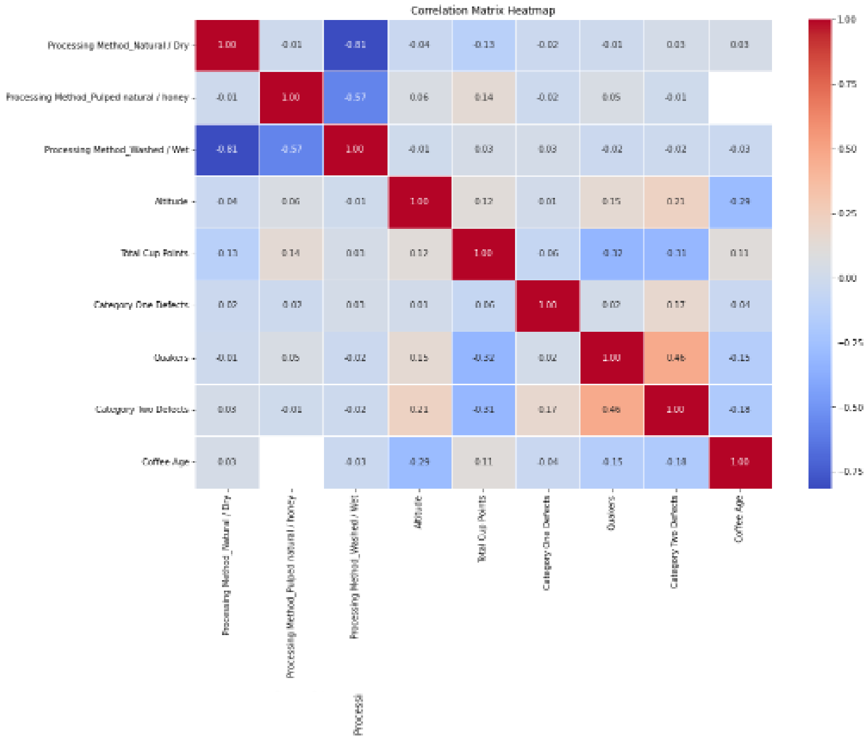
```
df_grouped = df.groupby('Country of Origin')['Total Cup Points'].mean().reset_index()

# Create a Choropleth map
fig = px.choropleth(df_grouped,
                    locations='Country of Origin',
                    locationmode='country names',
                    color='Total Cup Points',
                    hover_name='Country of Origin',
                    color_continuous_scale=px.colors.sequential.Plasma,
                    title='Average Total Cup Points by Country')

fig.show()

# Create a bar plot with gray color
plt.figure(figsize=(10, 5))
sns.barplot(x=df_grouped['Country of Origin'], y=df_grouped['Total Cup Points'], color='gray')
plt.title('Average Total Cup Points by Country')
plt.xlabel('Country of Origin')
plt.ylabel('Average Total Cup Points')
plt.xticks(rotation=90)
plt.show()
```

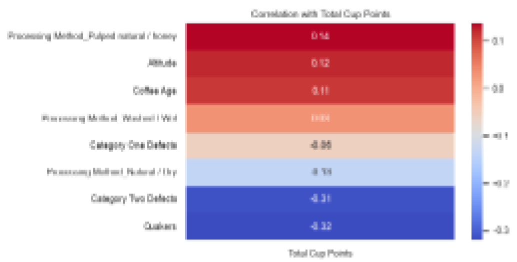
Average Total Cup Points by Country



This code calculates and visualizes the correlation matrix of the DataFrame processed_df. The first visualization is a heatmap showing the correlation coefficients between all pairs of columns, while the second visualization is a heatmap showing the correlations between each column and the 'Total Cup Points' column specifically.

```
correlation_matrix = processed_df.corr()

#Graph I.
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.2f')
plt.title("Correlation Matrix Heatmap")
plt.show()

corr = processed_df.corr()
target_corr = corr['Total Cup Points'].drop('Total Cup Points')

# Sort correlation values in descending order
target_corr_sorted = target_corr.sort_values(ascending=False)

#Graph II
# Create a heatmap of the correlations with the target column
sns.set(font_scale=0.8)
sns.set_style("white")
sns.set_palette("PuBuGn_d")
sns.heatmap(target_corr_sorted.to_frame(), cmap="coolwarm", annot=True, fmt='.2f')
plt.title('Correlation with Total Cup Points')
plt.show()
```





And the we got error.

We tried to do it again but we keep getting the same error:

TypeError: float() argument must be a string or a number, not 'datetime.datetime' or

AttributeError: Like 'float' object has no attribute 'timestamp'.

But there is no data in the dataset that we can convert accordingly.

In [26]:

```python
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.manifold import TSNE
import pandas as pd
import numpy as np
import plotly.express as px

# Create a copy of the dataframe to not alter the original
df_preprocessed = df.copy()

# Preprocessing: Label encoding for categorical variables
le = LabelEncoder()
categorical_features = ['Country of Origin', 'Processing Method', 'Color']
for feature in categorical_features:
    df_preprocessed[feature] = le.fit_transform(df[feature].astype(str))

# Preprocessing: MinMax scaling for numerical/ratio variables
mm = MinMaxScaler()
numerical_features = ['Altitude', 'Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Overall', 'Total Cup Points', 'Moisture Percentage', 'Category One Defects', 'Quakers', 'Category Two Defects', 'Coffee Age']
for feature in numerical_features:
    df_preprocessed[feature] = mm.fit_transform(df[feature].values.reshape(-1, 1))

# Apply t-SNE with different perplexity and learning rate
tsne = TSNE(n_components=2, random_state=42, perplexity=50, learning_rate=200)
tsne_results = tsne.fit_transform(df_preprocessed)

# Plotly Interactive plot
df_tsne = pd.DataFrame(data=tsne_results, columns=['Dim_1', 'Dim_2'])
df_tsne['Total Cup Points'] = df['Total Cup Points']
fig = px.scatter(df_tsne, x='Dim_1', y='Dim_2', color='Total Cup Points', title='t-SNE plot colored by Total Cup Points')
fig.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3800/1097278894.py in <module>
     18 numerical_features = ['Altitude', 'Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Overall', 'Total Cup Points', 'Moisture Percentage', 'Category One Defects', 'Quakers', 'Category Two Defects', 'Coffee Age']
     19 for feature in numerical_features:
---> 20     df_preprocessed[feature] = mm.fit_transform(df[feature].values.reshape(-1, 1))
     21 
     22 # Apply t-SNE with different perplexity and learning rate

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transformation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transformation)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in fit(self, X, y)
    361         # Reset internal state before fitting
    362         self._reset()
--> 363         return self.partial_fit(X, y)
    364 
    365     def partial_fit(self, X, y=None):

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\_data.py in partial_fit(self, X, y)
    394 
    395         first_pass = not hasattr(self, 'n_samples_seen_')
--> 396         X = self._validate_data(X, reset=first_pass,
    397                                 estimator=self, dtype=FLOAT_DTYPES,
    398                                 force_all_finite="allow-nan")

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    419             out = X
    420         elif isinstance(y, str) and y == 'no_validation':
--> 421             X = check_array(X, **check_params)
    422             out = X
    423         else:

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
```