In [ ]:
```
# for numerical computing
import numpy as np

# for dataframes
import pandas as pd

# for easier visualization
import seaborn as sns

# for visualization and to display plots
from matplotlib import pyplot as plt
%matplotlib inline

# import color maps
from matplotlib.colors import ListedColormap

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

from math import sqrt

# to split train and test set
from sklearn.model_selection import train_test_split

# to perform hyperparameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.linear_model import Ridge  # Linear Regression + L2 regularization
from sklearn.linear_model import Lasso  # Linear Regression + L1 regularization
from sklearn.svm import SVR # Support Vector Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Evaluation Metrics
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score as rs
from sklearn.metrics import mean_absolute_error as mae

#import xgboost
import os
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-7.2.0-posix-seh-rt_v5-rev0\\ming
w64\\bin'
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']
from xgboost import XGBRegressor
from xgboost import plot_importance  # to plot feature importance

# to save the final model on disk
from sklearn.externals import joblib
```

In [ ]:
```
np.set_printoptions(precision=2, suppress=True) #for printing floating point number
s upto  precision 2
```

Load real estate data from CSV

In [ ]:
```
df = pd.read_csv('BlackFriday 2.csv')
```

In [ ]:
```
df.shape
```

Columns of the dataset

In [ ]:
```
df.columns
```

Display the first 5 rows to see example observations.

In [ ]:
```
pd.set_option('display.max_columns', 20) ## display max 20 columns
df.head()
```

Some feaures are numeric and some are categorical

Filtering the categorical features:

In [ ]:
```
df.dtypes[df.dtypes=='object']
```

Distributions of numeric features

In [ ]:
```
# Plot histogram grid
df.hist(figsize=(16,16), xrot=-45) ## Display the labels rotated by 45 degress

# Clear the text "residue"
plt.show()
```

Obeservation:

The items in Category 1 were sold the most in the numbers of 2 and 5. Only around 2,000 costumers bought around 7 items belonging to Category 1.

Many buyers aren't married and many buyeres spend less money in category 1 then on 2 and 3.

Consider Product_Category_1, 2, 3: The items belonging to Product_Category_1 is the most popular category.

Consider Purchase: The highest number (around 113,000) of people made purchases of $5000 to $7000 while only about 2000 people bought around \$23,000 worth of items from the store.

Consider User_ID: On average, 50,000 Users were active shown by the number of their User_IDs.

Display summary statistics for categorical features.

In [ ]:
```
df.describe()
```

Distributions of categorical features

In [ ]:
```
df.describe(include=['object'])
```

Observation:

In [ ]:
```
#Almost 80% of the consumers were male:405380 out of 537577
#the most popular product was P00265242 with 1858 Sales.
#214,690 people of consumers were 26-35 of age group.
```

Bar plots for categorical Features

Plot bar plot for the 'Purchase' feature.

In [ ]:
```
plt.figure(figsize=(8,8))
sns.countplot(y='Gender', data=df)
```

Observations:

male are buying more and spending more in comparison to female.

The difference between purchases between male and female is huge which we can easily see in the above bar graph.

Similarly Plot bar plot for the 'City_Category' feature.

In [ ]:
```
plt.figure(figsize=(8,8))
sns.countplot(y='City_Category', data=df)
```

Segmentations

Segmentations are powerful ways to cut the data to observe the relationship between categorical features and numeric features.

Segmenting the target variable by key categorical features.

In [ ]:
```
sns.boxplot(y='Purchase', x='City_Category', data=df)
```

Observation:

In this case there are clearly some outliers in City A and B. city C spends more and purchase more in compared to City A and B.

In [ ]:
```
df.groupby('Gender').mean()
```

In [ ]:
```
# There is not a big difference between male and female in purchasing as male has s
lightly more purchasing.
# male are more employeed at difference of atleast 2.2
```

Boxplots

In [ ]:
```
sns.boxplot(y='City_Category', x='Purchase', data=df)
```

In [ ]:
```
# City of category A is considered the best location likewise B City is considered
second best and so on,
# Buyers of C category seems to spend a lot of money on average
#But B and A have some Buyers who spended alot of money which is not surprising bec
ause they might hav alot of money with them.
```

Segment by Gender and display the means and standard deviations within each class

In [ ]:
```
df.groupby('Gender').agg([np.mean, np.std])
```

Correlations

In [ ]:
```
# Finally, let's take a look at the relationships between numeric features and othe
r numeric features.
# Correlation is a value between -1 and 1 that represents how closely values for tw
o separate features.
# Positive correlation means that as one feature increases, the other increases.
# Negative correlation means that as one feature increases, the other decreases.
# Correlations near -1 or 1 indicate a strong relationship.
# Those closer to 0 indicate a weak relationship.
# 0 indicates no relationship.
```

In [ ]:
```
df.corr()
```

In [ ]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr())
```

In [ ]:
```
mask=np.zeros_like(df.corr())
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(10,10))
with sns.axes_style("white"):
    ax = sns.heatmap(df.corr()*100, mask=mask, fmt='.0f', annot=True, lw=1, cmap=Li
stedColormap(['green', 'yellow', 'red','blue']))
```

Data Cleaning

In [ ]:
```
# Dropping the duplicates (De-duplication)
```

In [ ]:
```
df = df.drop_duplicates()
print( df.shape )
```

In [ ]:
```
# It looks like we didn't have any duplicates in our original dataset.
# Even so, it's a good idea to check this as an easy first step for cleaning your d
ataset
```

Mislabeled Classes

In [ ]:
```
# Confirming if there is no negative classes in Stay_In_current_city_years
```

In [ ]:
```
sns.countplot(y='Stay_In_Current_City_Years', data=df)
```

Removing Outliers

Outliers can cause problems with certain types of models.

Boxplots are a nice way to detect outliers

Let's start with a box plot of your target variable, since that's what you're actually trying to predict

In [ ]:
```
sns.boxplot(df.Purchase)
```

Interpretation

In [ ]:
```
# The two vertical bars on the ends are the min and max values. All properties sold
for between \$200,000 and \$800,000.
# The box in the middle is the interquartile range (25th percentile to 75th percent
ile).
# Half of all observations fall in that box.
# Finally, the vertical bar in the middle of the box is the median.
```

In [ ]:
```
df.Purchase.sort_values(ascending=False).head()
```

In [ ]:
```
df = df[df.Purchase <= 15000]
df.shape
```

In [ ]:
```
## Plotting the boxplot of lot size after the change
sns.boxplot(df.Purchase)
```

Label missing categorical data

In [ ]:
```
# You cannot simply ignore missing values in your dataset.
# You must handle them in some way for the very practical reason that Scikit-Learn
algorithms
# do not accept missing values.
```

In [ ]:
```
# Display number of missing values by categorical feature
df.select_dtypes(include=['object']).isnull().sum()
```

In [ ]:
```
# so there are none and No values in any of the categorical data seem to be misssin
g
```

Flag and fill missing numeric data

In [ ]:
```
# Display number of missing values by numeric feature
df.select_dtypes(exclude=['object']).isnull().sum()
```

In [ ]:
```
# there are missing numerical data in category 2 and 3 as no purchase has been mad
e.
```

Before we move on to the next module, let's save the new dataframe we worked hard to clean.

In [ ]:
```
# This makes sure we don't have to re-do all the cleaning after clossing the sessio
n
```

In [ ]:
```
# Save cleaned dataframe to new file
df.to_csv(r'C:\Users\CHANDAN GURUNG\Desktop\cleaneddf.csv', index=False)
```

Feature Engineering

Indicator variables

In [ ]:
```
df['A_and_singles'] = ((df.City_Category == 'A') & (df.Marital_Status == 0)).astype
(int)
```

In [ ]:
```
# Display percent of rows where two_and_two == 1
df[df['A_and_singles']==1].shape[0]/df.shape[0]
```

Creating a new feature containing 10% tax on purchase

In [ ]:
```
df['Net_Tax'] = df.Purchase * 1.1
```

Machine Learning Models

Data Preparation

Train and Test Splits

In [ ]:
```
# Separate your dataframe into separate objects for the target variable (y)
# and the input features (X) and perform the train and test split
```

In [ ]:
```
# Create separate object for target variable
y = df.Purchase
# Create separate object for input features
X = df.drop('Purchase', axis=1)
```

In [ ]:
```
# Split X and y into train and test sets: 80-20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
te=1234)
```

In [ ]:
```
# Let's confirm we have the right number of observations in each subset
```

In [ ]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Data standardization

In [ ]:
```
# In Data Standardization we perform zero mean centring and unit scaling; i.e.
# we make the mean of all the features as zero and the standard deviation as 1.
# hus we use mean and standard deviation of each feature.
# It is very important to save the mean and standard deviation for each of the feat
ure from the training set,
# because we use the same mean and standard deviation in the test set.
```

In [ ]:
```
train_mean = X_train.mean()
train_std = X_train.std()
```

Standardize the train data set X_train = (X_train - train_mean) / train_std

In [ ]:
```
## Check for mean and std dev.
X_train.describe()
```

In [ ]:
```
## Note: We use train_mean and train_std_dev to standardize test data set
X_test = (X_test - train_mean) / train_std
```

In [ ]:
```
## Check for mean and std dev. - not exactly 0 and 1
X_test.describe()
```

Model 1 - Baseline Model

In [ ]:
```
# In this model, for every test data point, we will simply predict the average of t
he train labels as the output.
# We will use this simple model to perform hypothesis testing for other complex mod
els.
```

In [ ]:
```
## Predict Train results
y_train_pred = np.ones(y_train.shape[0])*y_train.mean()
```

In [ ]:
```
# Predict Test results
y_pred = np.ones(y_test.shape[0])*y_train.mean()
from sklearn.metrics import r2_score
```

In [ ]:
```
print("Train Results for Baseline Model:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
print("Mean Absolute Error: ", mae(y_train.values, y_train_pred))
```

In [ ]:
```
print("Results for Baseline Model:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
print("Mean Absolute Error: ", mae(y_test, y_pred))
```

Storing all the datasets

In [ ]:
```
win_model = RandomForestRegressor(n_estimators=200, min_samples_split=10, min_sampl
es_leaf=2)
win_model.fit(X_train, y_train)
with open('rfr_real_estate.pkl', 'wb') as pickle_file:
       joblib.dump(win_model, 'rfr_real_estate.pkl')
```

In [ ]: