Weekly Assignment 6

Brandon Owens and Loan Pham

Q. 1

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

In [2]:
```
# Import the dataset “iowa_housing.csv”. The dataset is about housing prices in Am
# You are going to predict sale price.
df = pd.read_csv("../dataFiles/iowa_housing.csv")
df.head()
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	.
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

5 rows × 81 columns

In [3]:
```
# (a)   Show the head and shape of the dataframe. Drop “Id” in the dataset.
df_drop = df.drop(["Id"], axis=1)
df_drop.head()
```

Out[3]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotC
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	I
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	I
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	C

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotC
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

5 rows × 80 columns

In [4]:
```
df_drop.shape
```

Out[4]: (1460, 80)

In [5]:
```
# (b)   Use the dataset to:
# (i)   create a correlation matrix (df.corr()), and then draw the heatmap on the
corr = df_drop.corr()
corr.head()
```

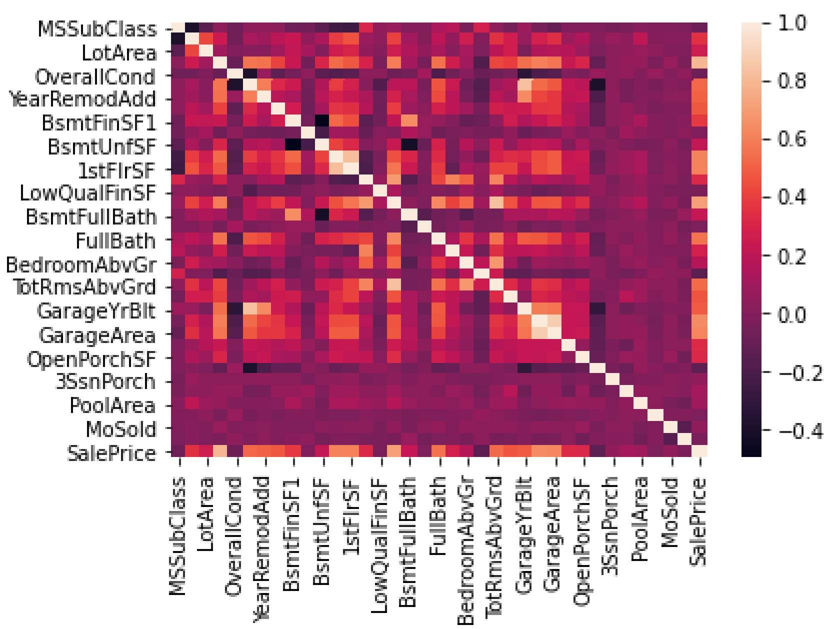
Out[5]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
MSSubClass	1.000000	-0.386347	-0.139781	0.032628	-0.059316	0.027850	0.040581
LotFrontage	-0.386347	1.000000	0.426095	0.251646	-0.059213	0.123349	0.088866
LotArea	-0.139781	0.426095	1.000000	0.105806	-0.005636	0.014228	0.013788
OverallQual	0.032628	0.251646	0.105806	1.000000	-0.091932	0.572323	0.550684
OverallCond	-0.059316	-0.059213	-0.005636	-0.091932	1.000000	-0.375983	0.073741

5 rows × 37 columns

In [6]:
```
sns.heatmap(corr)
```

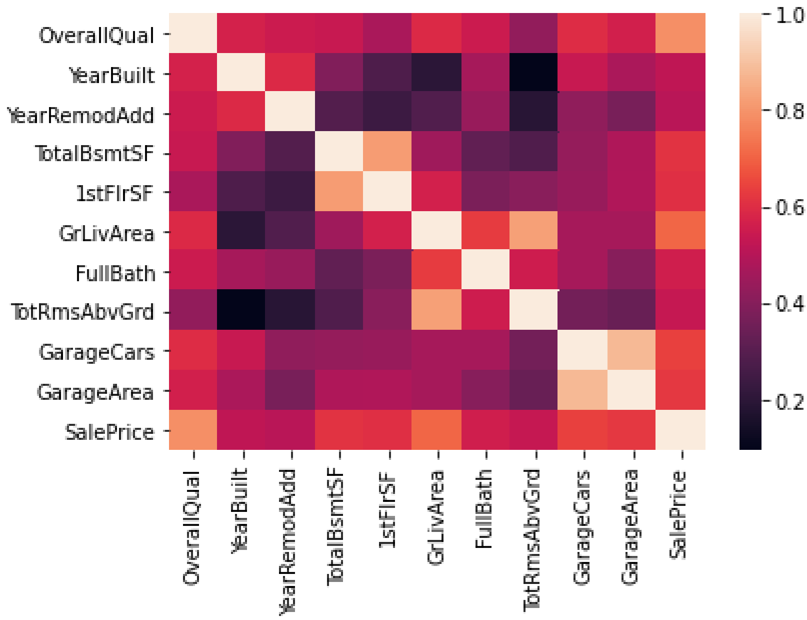
Out[6]: <AxesSubplot:>



In [7]:

```
# (ii)  Find the features that the absolute value of correlation with “SalePrice”
# Then, draw the heatmap of correlation on those features.
features_abs = corr[abs((corr.SalePrice)>=.5)].SalePrice.keys()
abs_corr = corr.loc[features_abs,features_abs]
sns.heatmap(abs_corr)
```

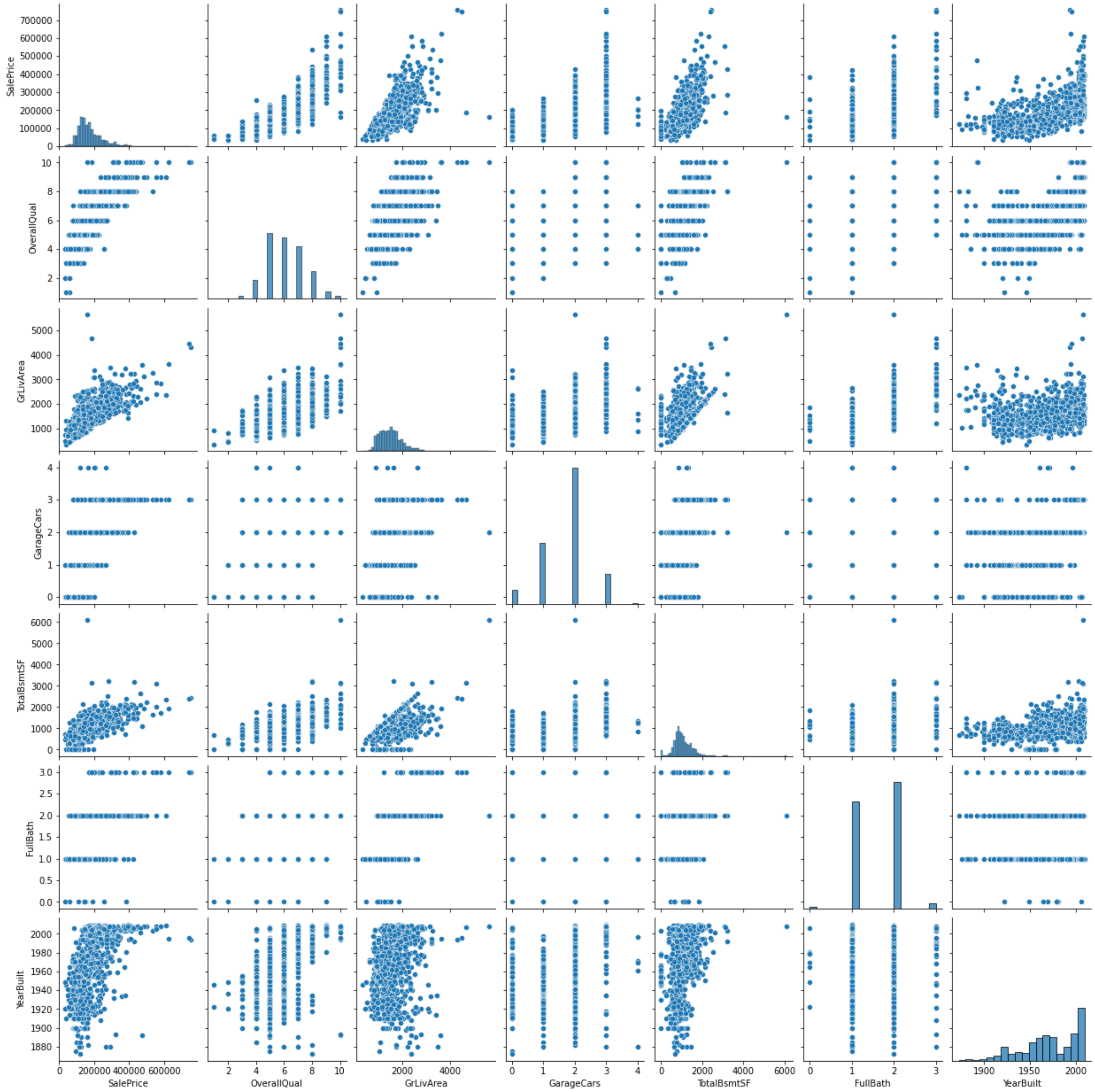
Out[7]: `<AxesSubplot:>`



In [8]:

```
# (iii) create pair plots on the following features
sns.pairplot(df_drop, vars=['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars',
```

Out[8]: `<seaborn.axisgrid.PairGrid at 0x22200026ac0>`



In [9]:

```
# (iv)  Find the most important feature relative to the “SalePrice” based on absol
corr.sort_values(["SalePrice"], ascending = False, inplace = True)
corr.SalePrice.head()
```

Out[9]:

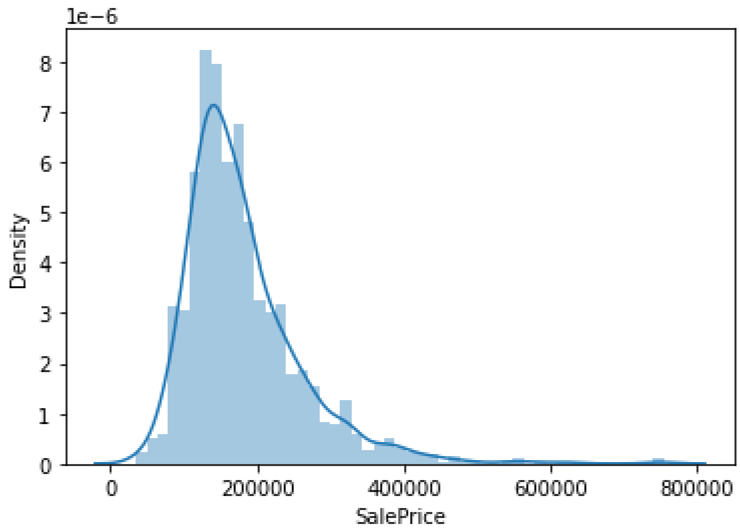
```
SalePrice      1.000000
OverallQual    0.790982
GrLivArea      0.708624
GarageCars     0.640409
GarageArea     0.623431
Name: SalePrice, dtype: float64
```

In [10]:

```
# (v)   create a distribution plot on “SalePrice”
sns.distplot(df_drop['SalePrice'], kde =True)
```

```
C:\Users\jeric\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]: <AxesSubplot:xlabel='SalePrice', ylabel='Density'>



In [11]:

```
# (vi)  For all the numerical features except “SalePrice”,
# replace all the missing values of numerical features with the median value of ea
temp=pd.DataFrame()
temp['SalePrice'] = df['SalePrice']
num_cols = df.select_dtypes(exclude = ['object']).columns
num_cols = num_cols.drop('SalePrice')
med_replace = df[num_cols]
med_replace = med_replace.fillna(med_replace.median())
med_replace
```

Out[11]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	M
0	1	60	65.0	8450	7	5	2003	2003	
1	2	20	80.0	9600	6	8	1976	1976	
2	3	60	68.0	11250	7	5	2001	2002	
3	4	70	60.0	9550	7	5	1915	1970	
4	5	60	84.0	14260	8	5	2000	2000	
...	...	...	...	...	...	...	...	...	
1455	1456	60	62.0	7917	6	5	1999	2000	
1456	1457	20	85.0	13175	6	6	1978	1988	
1457	1458	70	66.0	9042	7	9	1941	2006	
1458	1459	20	68.0	9717	5	6	1950	1996	
1459	1460	20	75.0	9937	5	6	1965	1965	

1460 rows × 37 columns

In [12]:

```
# (vii) Create dummies for all categorical features.
# The final shape of dataset should be (1460, 246) (set drop_first = True)
dummies_cols = df.select_dtypes(include = ['object']).columns
```

```
new_housing = df[dummies_cols]
new_housing = pd.get_dummies(new_housing, drop_first = True)
new_housing.head()
```

Out[12]:

	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave	Alley_Pave	LotShape_IR2
0	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0
2	0	0	1	0	1	0	0
3	0	0	1	0	1	0	0
4	0	0	1	0	1	0	0

5 rows × 209 columns

In [13]:

```
newdf_housing = pd.concat([temp, med_replace, new_housing], axis =1)
newdf_housing.shape
```

Out[13]:

```
(1460, 247)
```

In [14]:

```
newdf_housing['SalePrice']
```

Out[14]:

```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

In [15]:

```
#  (c) Then, do the modelling:
```

In [16]:

```
# (i)   Check for any missing values again.
newdf_housing.isnull().any()
```

Out[16]:

```
SalePrice                False
Id                       False
MSSubClass               False
LotFrontage              False
LotArea                  False
                         ...
SaleCondition_AdjLand    False
SaleCondition_Alloca     False
SaleCondition_Family     False
SaleCondition_Normal     False
SaleCondition_Partial    False
Length: 247, dtype: bool
```

In [17]:
```
# (ii)  create y  = “SalePrice”. Drop y from the dataframe.
y = "SalePrice"
house_temp = newdf_housing.drop([y], axis =1)
house_temp.head()
```

Out[17]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnr
0	1	60	65.0	8450	7	5	2003	2003	
1	2	20	80.0	9600	6	8	1976	1976	
2	3	60	68.0	11250	7	5	2001	2002	
3	4	70	60.0	9550	7	5	1915	1970	
4	5	60	84.0	14260	8	5	2000	2000	

5 rows × 246 columns

In [18]:
```
# (iii) Split the dataset into train and test.

X_train, X_test, y_train, y_test = train_test_split(house_temp,newdf_housing, test
print("X_train : " + str(X_train.shape))
print("X_test : " + str(X_test.shape))
print("y_train : " + str(y_train.shape))
print("y_test : " + str(y_test.shape))
```

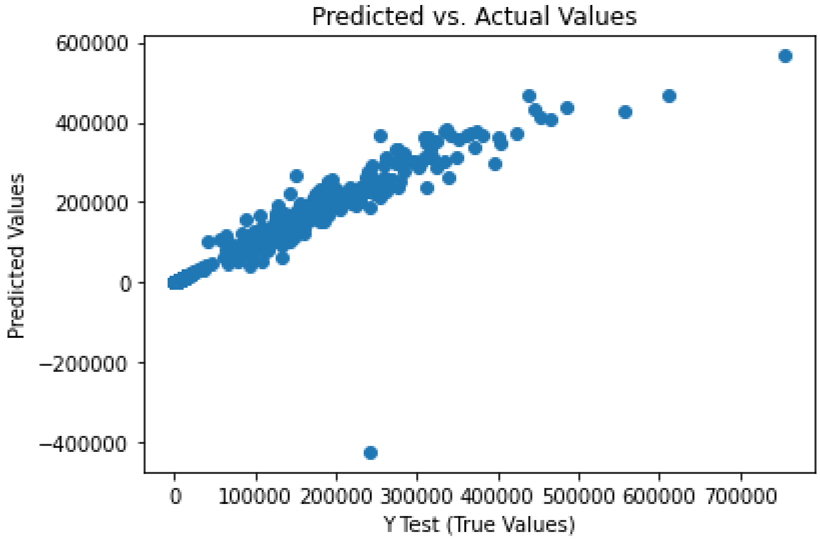
```
X_train : (1022, 246)
X_test : (438, 246)
y_train : (1022, 247)
y_test : (438, 247)
```

In [19]:
```
# (iv)  Train a linear regression model.
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[19]:
```
LinearRegression()
```

In [20]:
```
# (v)   predict “SalePrice” with the test data.
y_pred = model.predict(X_test)
plt.scatter(y_test,y_pred)
plt.xlabel('Y Test (True Values)')
plt.ylabel('Predicted Values')
plt.title('Predicted vs. Actual Values ')
```

Out[20]:
```
Text(0.5, 1.0, 'Predicted vs. Actual Values ')
```



In [21]:

```
# (vi)  find the root mean squared error of the model on the test data.
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Root Mean Squared Error = ", rmse)
print("The range of temperature = ",y_test.min(), y_test.max())
```

```
Root Mean Squared Error =  2681.0955819197743
The range of temperature =  SalePrice                35311.0
Id                         11.0
MSSubClass                 20.0
LotFrontage                21.0
LotArea                  1491.0
                          ...
SaleCondition_AdjLand       0.0
SaleCondition_Alloca        0.0
SaleCondition_Family        0.0
SaleCondition_Normal        0.0
SaleCondition_Partial       0.0
Length: 247, dtype: float64 SalePrice                755000.0
Id                        1455.0
MSSubClass                 190.0
LotFrontage                174.0
LotArea                  70761.0
                           ...
SaleCondition_AdjLand        0.0
SaleCondition_Alloca         1.0
SaleCondition_Family         1.0
SaleCondition_Normal         1.0
SaleCondition_Partial        1.0
Length: 247, dtype: float64
```

Q2

In [22]:

```
# (a)   Read in the dataset 'online_shoppers_intention.csv'
df_shoppers = pd.read_csv('../dataFiles/online_shoppers_intention.csv')
df_shoppers.head()
```

Out[22]:

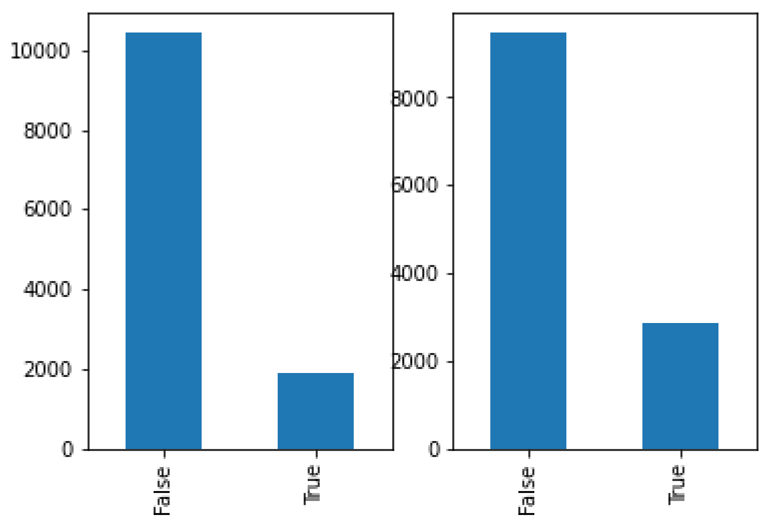
	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	Prod
0	0.0	0.0	0.0	0.0	1.0	

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	Prod
1	0.0	0.0	0.0	0.0	2.0	
2	0.0	-1.0	0.0	-1.0	1.0	
3	0.0	0.0	0.0	0.0	2.0	
4	0.0	0.0	0.0	0.0	10.0	

In [23]:

```
# (b)   Count how many shoppers buy, i.e. “Revenue”==True.
# Count how many shoppers browse in the weekends, i.e. “Weekends”==True. Create th
fig, ax =plt.subplots(1,2)
df_shoppers['Revenue'].value_counts().plot.bar(ax=ax[0])
df_shoppers['Weekend'].value_counts().plot.bar(ax=ax[1])
```

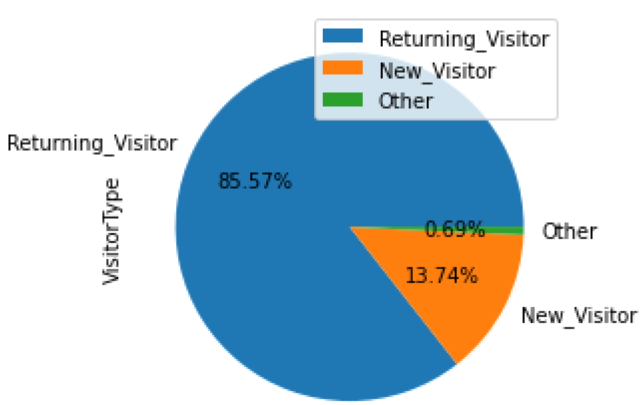
Out[23]: <AxesSubplot:>



In [24]:

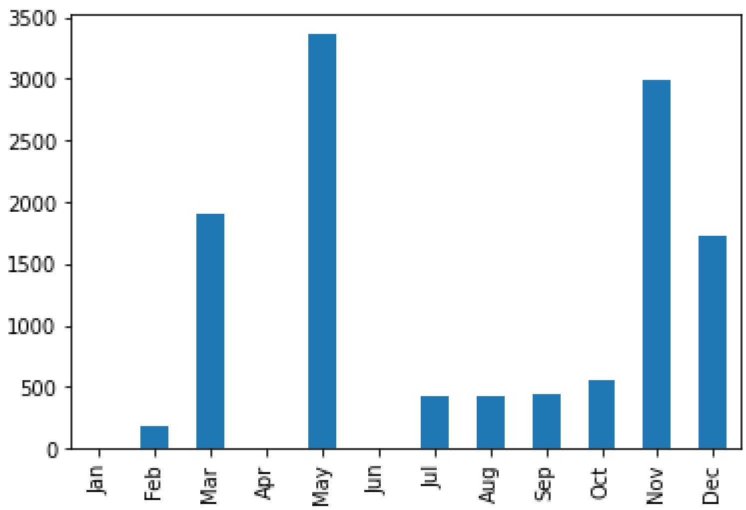
```
# (c)   Create the following plot, which shows the proportions of various kinds of
# (first, use value counts, and then use  .plot.pie(autopct = '%.2f%%')
df_shoppers['VisitorType'].value_counts().plot.pie(autopct = '%.2f%%')
plt.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x22202a77370>



In [25]:
```
# (d)   Check the month with most shoppers visiting the online shopping sites. Cre
months = ['Jan', 'Feb', 'Mar', 'Apr','May','Jun', 'Jul', 'Aug','Sep', 'Oct', 'Nov'
new_df = df_shoppers['Month'].value_counts()
new_df = new_df.reindex(months)
new_df.plot.bar()
```

Out[25]: <AxesSubplot:>



In [26]:
```
# (e)   For all the numerical variables, fill the missing values with the median c
num_features = df_shoppers.select_dtypes(exclude = ["object"]).columns
num_shoppers = df_shoppers[num_features]
num_shoppers = num_shoppers.fillna(num_shoppers.median())
num_shoppers
```

Out[26]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	2.0
2	0.0	-1.0	0.0	-1.0	1.0
3	0.0	0.0	0.0	0.0	2.0
4	0.0	0.0	0.0	0.0	10.0
...	...	...	...	...	...
12325	3.0	145.0	0.0	0.0	53.0
12326	0.0	0.0	0.0	0.0	5.0
12327	0.0	0.0	0.0	0.0	6.0
12328	4.0	75.0	0.0	0.0	15.0
12329	0.0	0.0	0.0	0.0	3.0

12330 rows × 16 columns

In [27]:
```
# (f)   Create dummies for categorical variables.
```

```
cate_vari = df_shoppers.select_dtypes(include = ["object"]).columns
shop_cat = df_shoppers[cate_vari]
shop_cat = pd.get_dummies(shop_cat,drop_first = True)
shop_cat.head()
```

Out[27]:

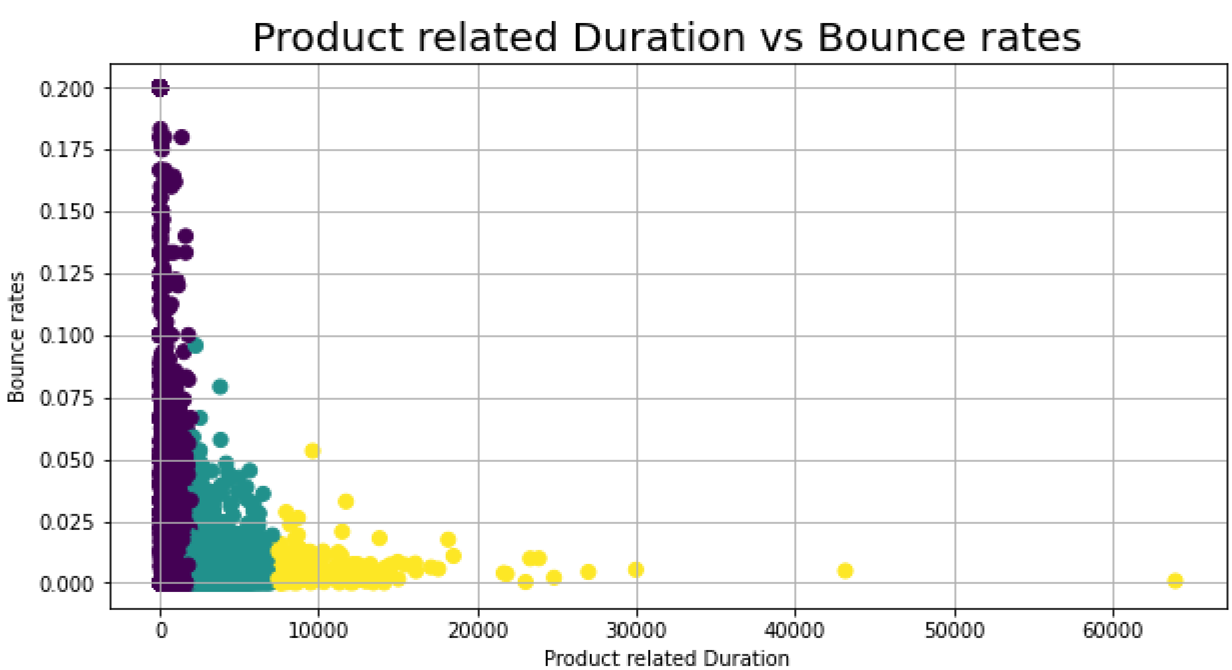
	Month_Dec	Month_Feb	Month_Jul	Month_June	Month_Mar	Month_May	Month_Nov	Month_Oct
0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0

In [28]:

```
# (g)   Use KMeans to group customers into 3 clusters. That is unsupervised learni

kmeans = KMeans(n_clusters=3)
kmeans.fit(num_shoppers)
y_kmeans = kmeans.predict(num_shoppers)
plt.figure(figsize=(10,5))
plt.title('Product related Duration vs Bounce rates', fontsize = 20)
plt.grid()
plt.xlabel('Product related Duration')
plt.ylabel('Bounce rates')
plt.scatter(df_shoppers["ProductRelated_Duration"], df_shoppers["BounceRates"], c=
```

Out[28]: `<matplotlib.collections.PathCollection at 0x222056b79d0>`



In [29]:

```
# (h)   Set y = "Revenue" and X is the dataframe without "Revenue".
numerical_cols = df_shoppers.select_dtypes(exclude = ["number"]).columns
shop_nums = df_shoppers[numerical_cols]
df_dummies = pd.get_dummies(shop_nums, drop_first=True)
y = df_dummies["Revenue"]
```

```
X = df_dummies.drop("Revenue", axis=1)
df_dummies.isnull().any()
```

Out[29]:
```
Weekend                           False
Revenue                           False
Month_Dec                         False
Month_Feb                         False
Month_Jul                         False
Month_June                        False
Month_Mar                         False
Month_May                         False
Month_Nov                         False
Month_Oct                         False
Month_Sep                         False
VisitorType_Other                 False
VisitorType_Returning_Visitor     False
dtype: bool
```

In [30]:
```
# (i)   Split the dataset into train and test sets.
# Train a support vector machine classifier model. Use kernel = "rbf" and class_we
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_s
model = SVC(kernel='rbf', class_weight='balanced')
model.fit(X_train, y_train)
```

Out[30]:
```
SVC(class_weight='balanced')
```

In [31]:
```
# (j)   Predict which online shopper will do a purchase. Find the accuracy score.
y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[31]:
```
0.6387318751536004
```

Q3

In [32]:
```
# (a)   Load the dataset.
from sklearn.datasets import fetch_olivetti_faces
face= fetch_olivetti_faces()
faces = fetch_olivetti_faces().images
```

In [33]:
```
# (b)   Show the first 10 images
fig, ax = plt.subplots(1, 10, figsize=(64, 64))
for i, axi in enumerate(ax.flat):
    axi.imshow(faces[i], cmap=plt.cm.bone)
    axi.set(xticks=[], yticks=[])
```

In [34]:
```
# (c)   Size of each image is 64x64. Use PCA to reduce it into 90 features.
# For the first row, show the first 10 original images.
# Then for the second row, show the first 10 images with reduced number of feature
x = face.data
```

```
y = face.target
pca = PCA(n_components = 90)
pca.fit(x)
transformed_data = pca.fit_transform(x)
x_approx = pca.inverse_transform(transformed_data)
x_approx_img = x_approx.reshape(400,64,64)

fig, ax = plt.subplots(1, 10, figsize=(64, 64))
for i, axi in enumerate(ax.flat):
    axi.imshow(faces[i], cmap=plt.cm.bone)
    axi.set(xticks=[], yticks=[])

fig, ax = plt.subplots(1, 10, figsize=(64, 64))
for i, axi in enumerate(ax.flat):
    axi.imshow(x_approx_img[i] , cmap = plt.cm.bone)
    axi.set(xticks=[], yticks=[])
```

In [35]:

```
# (d)   Using images of reduced features to conduct the machine learning task.
# (i)   Split the dataset into train and test.

X_train, X_test, y_train, y_test = train_test_split(face.data,face.target, random_
```

In [36]:

```
pca = PCA(n_components=90)
pca.fit(X_train)
```

Out[36]: PCA(n_components=90)

In [37]:

```
# (ii)  Train a Random Forest Classifier model. Set n_estimators=100. Predict the

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [38]:

```
# (iii) Find the accuracy score.
accuracy_score(y_test,y_pred)
```

Out[38]: 0.95

In [39]:

```
# (iv)  Create a confusion matrix, and put it in a heatmap. (Your heatmap may look
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [40]:

```
sns.heatmap(cm)
```

Out[40]: <AxesSubplot:>