AIRBNB Price Prediction in Newyork City

In this kernel we are focusing Price Prediction of New York City Airbnb Open Data Airbnb listings and metrics in NYC, NY, USA (2019)for linear regression.

Data Since 2008, guests and hosts have used Airbnb to expand on traveling possibilities and present more unique, personalized way of experiencing the world. This dataset describes the listing activity and metrics in NYC, NY for 2019. This data file includes all needed information to find out more about hosts, geographical availability, necessary metrics to make predictions and draw conclusions.

This data contains 16 columns, 4852 unique values(samples). Imported all necessary files and libraries, We removed unnecessary data from the datset like last review, reviews per month and host name as they donot support the data required. We filled the null values with zero constant and did the visualization using seaborn, pyplot, matplotlib.

Variables id: listing ID name: name of the listing host_id: host ID host_name: name of the host neighbourhood_group: location neighbourhood: area latitude: latitude coordinateslatitude: latitude coordinates longitude: longitude coordinates room_type: listing space type price: price in dollars minimum_nights: amount of nights minimum number_of_reviews: number of reviews last_review: latest review reviews_per_month: number of reviews per month calculated_host_listings_count: amount of listing per host availability_365: number of days when listing is available for booking

We will perform Regression on this dataset.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns

from scipy.stats import norm
from scipy import stats
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn import metrics

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
my_data = pd.read_csv('NYC_AirBNB.csv')
```

In [3]:

```
my_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4852 entries, 0 to 4851
Data columns (total 30 columns):
Unnamed: 0                4852 non-null int64
id                        4852 non-null int64
log_price                 4852 non-null float64
property_type             4852 non-null object
room_type                 4852 non-null object
amenities                 4852 non-null object
accommodates              4852 non-null int64
bathrooms                 4838 non-null float64
bed_type                  4852 non-null object
```

```
cancellation_policy        4852 non-null object
cleaning_fee               4852 non-null object
city                       4852 non-null object
description                4852 non-null object
first_review               3833 non-null object
host_has_profile_pic       4814 non-null object
host_identity_verified     4814 non-null object
host_response_rate         3348 non-null object
host_since                 4814 non-null object
instant_bookable           4852 non-null object
last_review                3837 non-null object
latitude                   4852 non-null float64
longitude                  4852 non-null float64
name                       4852 non-null object
neighbourhood              4852 non-null object
number_of_reviews          4852 non-null int64
review_scores_rating       3759 non-null float64
thumbnail_url              4494 non-null object
zipcode                    4789 non-null object
bedrooms                   4850 non-null float64
beds                       4840 non-null float64
dtypes: float64(7), int64(4), object(19)
memory usage: 1.1+ MB
```

In [4]:

```
my_data.head()
```

Out[4]:

	Unnamed: 0	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_
0	16553	44472	4.382027	Condominium	Private room	{TV,"Cable TV",Internet,"Wireless Internet","A...	1	1.0	Real Bed	
1	12555	7007348	5.075174	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	3	1.0	Real Bed	
2	15012	10283574	4.852030	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	2	1.0	Real Bed	mo
3	21502	1754527	5.010635	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	4	1.0	Real Bed	
4	13431	16823953	4.317488	Apartment	Private room	{TV,"Wireless Internet","Air conditioning",Kit...	2	1.0	Real Bed	

5 rows × 30 columns

In [5]:

```
def calculate_nullvalue(df):
    '''
        Calculating  percentage of null value present in our data set.
    '''

    missing_value = df.isnull().sum().to_frame()
    missing_value.columns = ['num_nullvalue']
    missing_value['percetage_missing'] = np.round(100 * (missing_value['num_nullvalue'] / df.shape[
0]))
    missing_value.sort_values(by='num_nullvalue', ascending=False, inplace=True)

    return missing_value
```

In [6]:

```
num_nullvalue = calculate_nullvalue(my_data)
num_nullvalue
```

```
num_nullvalue
```

Out[6]:

	num_nullvalue	percetage_missing
host_response_rate	1504	31.0
review_scores_rating	1093	23.0
first_review	1019	21.0
last_review	1015	21.0
thumbnail_url	358	7.0
zipcode	63	1.0
host_identity_verified	38	1.0
host_since	38	1.0
host_has_profile_pic	38	1.0
bathrooms	14	0.0
beds	12	0.0
bedrooms	2	0.0
city	0	0.0
longitude	0	0.0
log_price	0	0.0
property_type	0	0.0
room_type	0	0.0
number_of_reviews	0	0.0
neighbourhood	0	0.0
name	0	0.0
latitude	0	0.0
description	0	0.0
amenities	0	0.0
instant_bookable	0	0.0
accommodates	0	0.0
bed_type	0	0.0
id	0	0.0
cancellation_policy	0	0.0
cleaning_fee	0	0.0
Unnamed: 0	0	0.0

In [7]:

```
#Finding the missing values in the dataframe
my_data.isnull().sum()
```

Out[7]:

```
Unnamed: 0                 0
id                         0
log_price                  0
property_type              0
room_type                  0
amenities                  0
accommodates               0
bathrooms                 14
bed_type                   0
cancellation_policy        0
cleaning_fee               0
city                       0
description                0
first_review            1019
host_has_profile_pic      38
host_identity_verified    38
```

```
host_response_rate        1504
host_since                  38
instant_bookable             0
last_review               1015
latitude                     0
longitude                    0
name                         0
neighbourhood                0
number_of_reviews            0
review_scores_rating      1093
thumbnail_url              358
zipcode                     63
bedrooms                     2
beds                        12
dtype: int64
```

In [8]:

```
Conversion = {'Condo':['Timeshare','Loft','Guest suite','Condominium','Serviced apartment'],
         'Housing':['Vacation home','Townhouse','Casa particular','Villa','In-law'],
         'Hotel type 1':['Dorm','Guesthouse','Hostel'],
         'Hotel type 2':['Bed & Breakfast','Boutique hotel'],
         'Other':['Island','Yurt','Hut','Treehouse',
                  'Earth House','Tipi','Train','Parking Space','Lighthouse',
                 'Cabin','Camper/RV','Bungalow','Cave','Castle','Chalet','Boat','Tent']
        }
Conversion_real = {i : k for k, v in Conversion.items() for i in v}
my_data['property_type'].replace(Conversion_real,inplace =True)
```

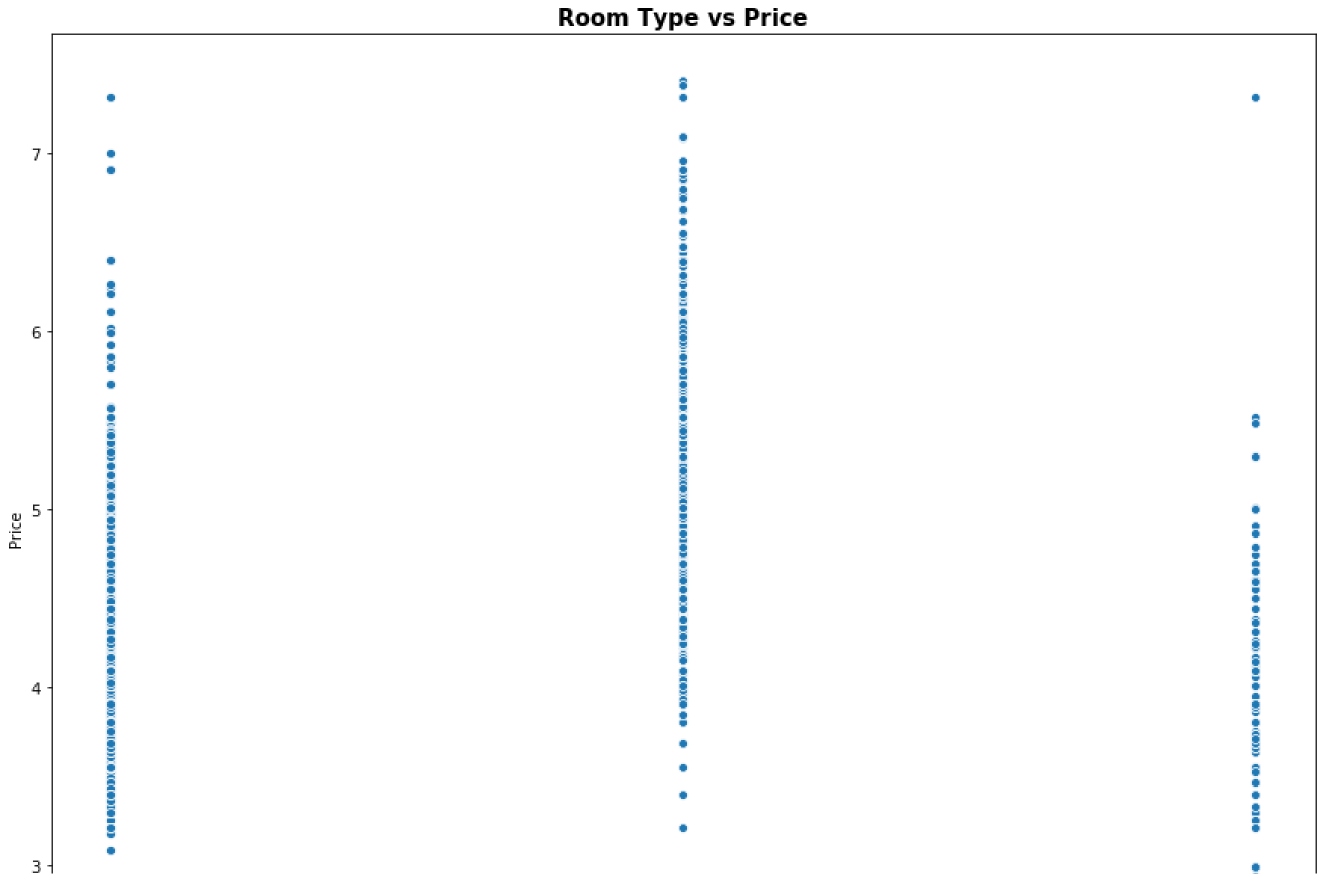
In [9]:

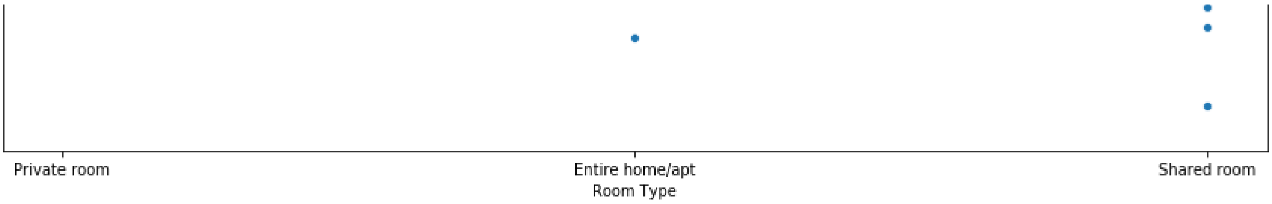
```
plt.figure(figsize=(15,12))
sns.scatterplot(x='room_type', y='log_price', data=my_data)

plt.xlabel("Room Type", size=10)
plt.ylabel("Price", size=10)
plt.title("Room Type vs Price",size=15, weight='bold')
```

Out[9]:

```
Text(0.5, 1.0, 'Room Type vs Price')
```





In [10]:

```
top10_freq_neighbourhood=my_data.neighbourhood.value_counts().head(10)
print(top10_freq_neighbourhood)
```

```
Williamsburg          442
Bedford-Stuyvesant    296
Bushwick              248
Upper West Side       210
Harlem                208
Upper East Side       185
Crown Heights         184
Hell's Kitchen        178
Lower East Side       137
East Harlem           137
Name: neighbourhood, dtype: int64
```

In [11]:

```
top10_freq_neighbourhood_data=my_data[my_data['neighbourhood'].isin(['Williamsburg','Bedford-Stuyv
esant','Harlem','Bushwick',
'Upper West Side','Hell\'s Kitchen','East Village','Upper East Side','Crown Heights','Midtown'])]
top10_freq_neighbourhood_data
```

Out[11]:

	Unnamed: 0	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_typ
3	21502	1754527	5.010635	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	4	1.0	Real B
4	13431	16823953	4.317488	Apartment	Private room	{TV,"Wireless Internet","Air conditioning",Kit...	2	1.0	Real B
6	5389	2636988	4.262680	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Heatin...	2	1.0	Real B
7	21857	20965965	4.248495	Apartment	Private room	{TV,Internet,"Wireless Internet","Air conditio...	2	1.0	Real B
10	29298	2182851	3.806662	Apartment	Private room	{"Pets live on this property",Cat(s),"Smoke de...	2	1.0	Real B
...	...	...	...	...	...	...	...	...	
4842	24714	21023168	4.867534	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	5	1.0	Real B
4843	8047	11145641	4.276666	Apartment	Private room	{TV,Internet,"Wireless Internet","Air conditio...	2	1.0	Real B
4845	17804	14139651	3.688879	Apartment	Private room	{"Wireless Internet",Kitchen,Heating,Essentials}	2	1.0	Real B
4846	5608	12524258	4.094345	Apartment	Private room	{Internet,"Wireless Internet",Kitchen,Heating,...	4	1.0	Real B

	Unnamed: 0	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type
4848	20544	19120020	[illegible]	Apartment	Private room	{"Wireless Internet",Kitchen,Elevator,Heating,...	1	1.0	Real Bed

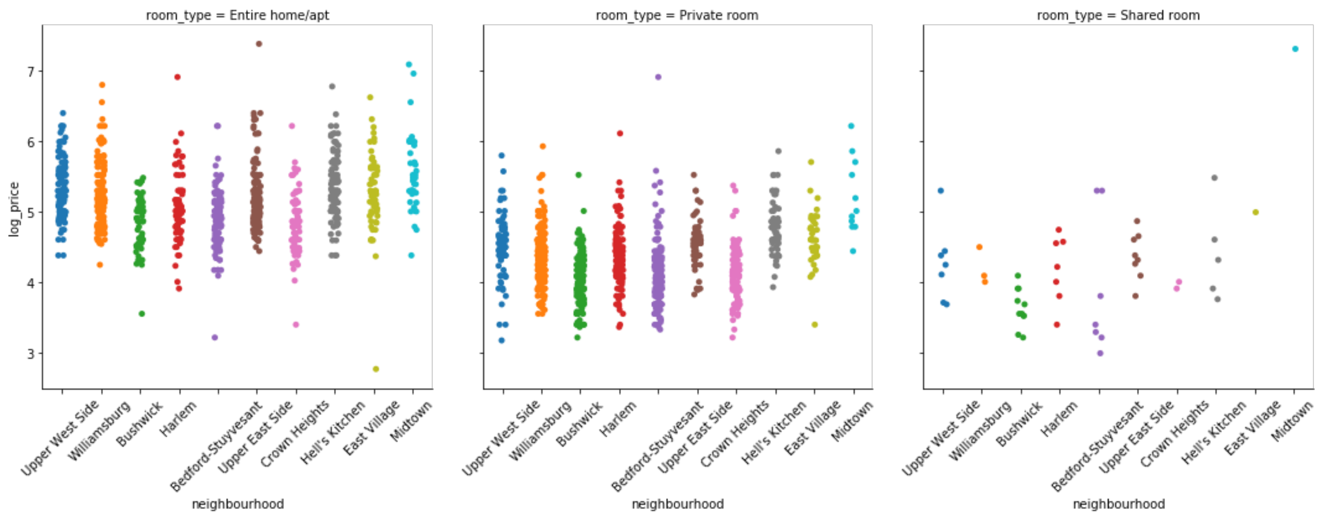
2111 rows × 30 columns

In [12]:

```
t=sns.catplot(x="neighbourhood", y="log_price", col="room_type", data=top10_freq_neighbourhood_data
)
t.set_xticklabels(rotation=45)
```

Out[12]:

```
<seaborn.axisgrid.FacetGrid at 0x294aee9bb48>
```

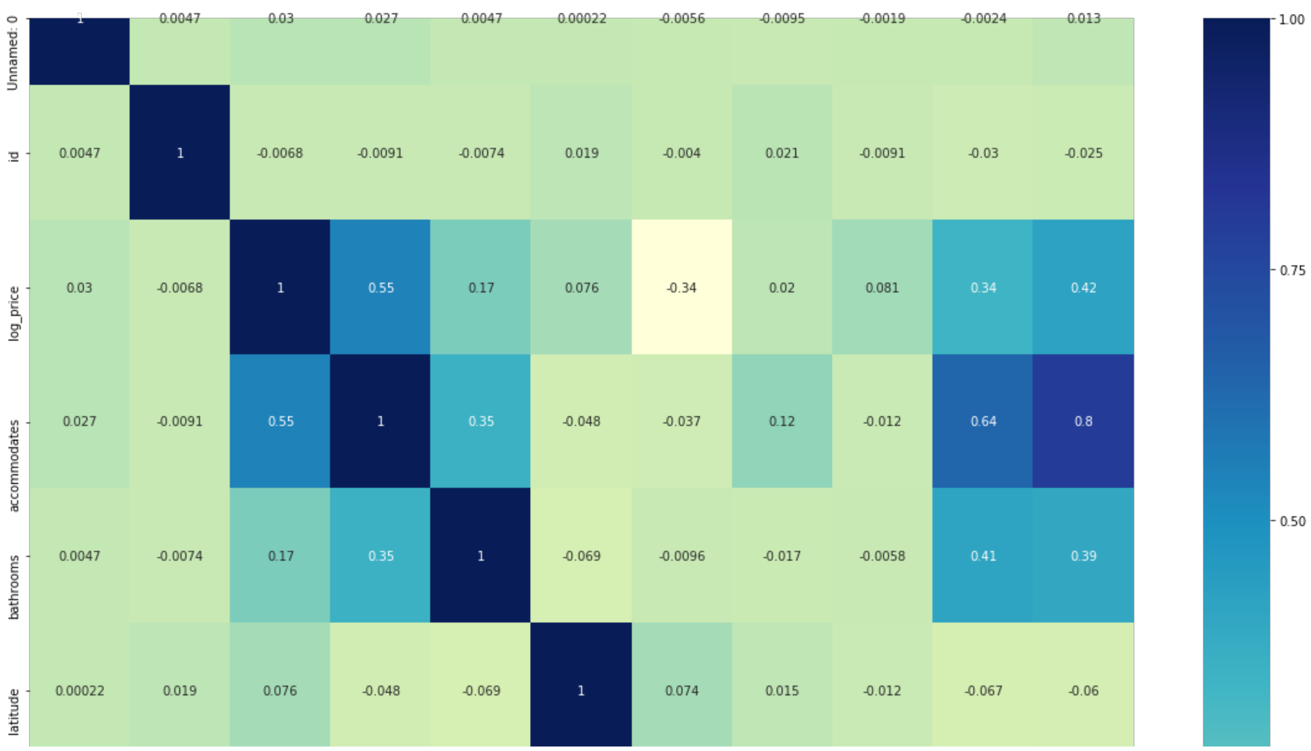


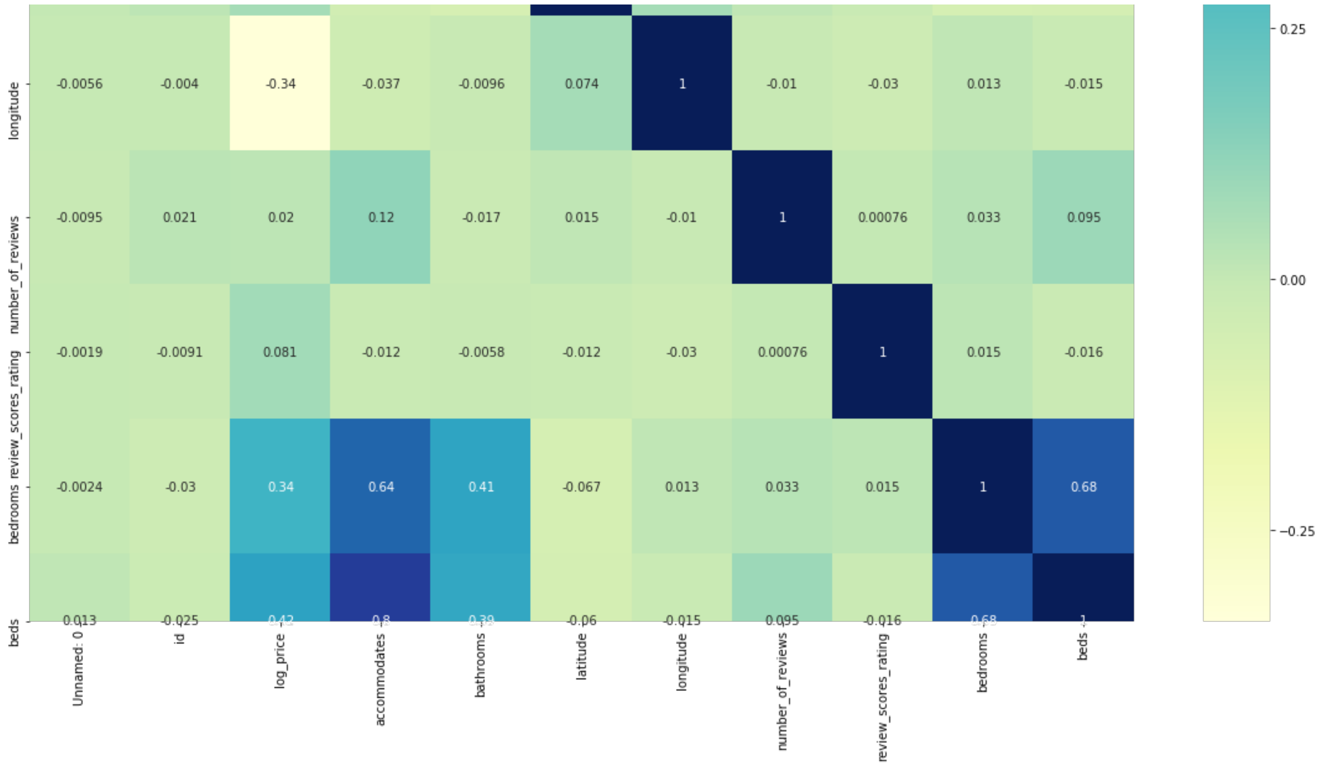
In [13]:

```
plt.figure(figsize=(20,20))
sns.heatmap(my_data.corr(), annot=True, cmap="YlGnBu")
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x294af20af48>
```



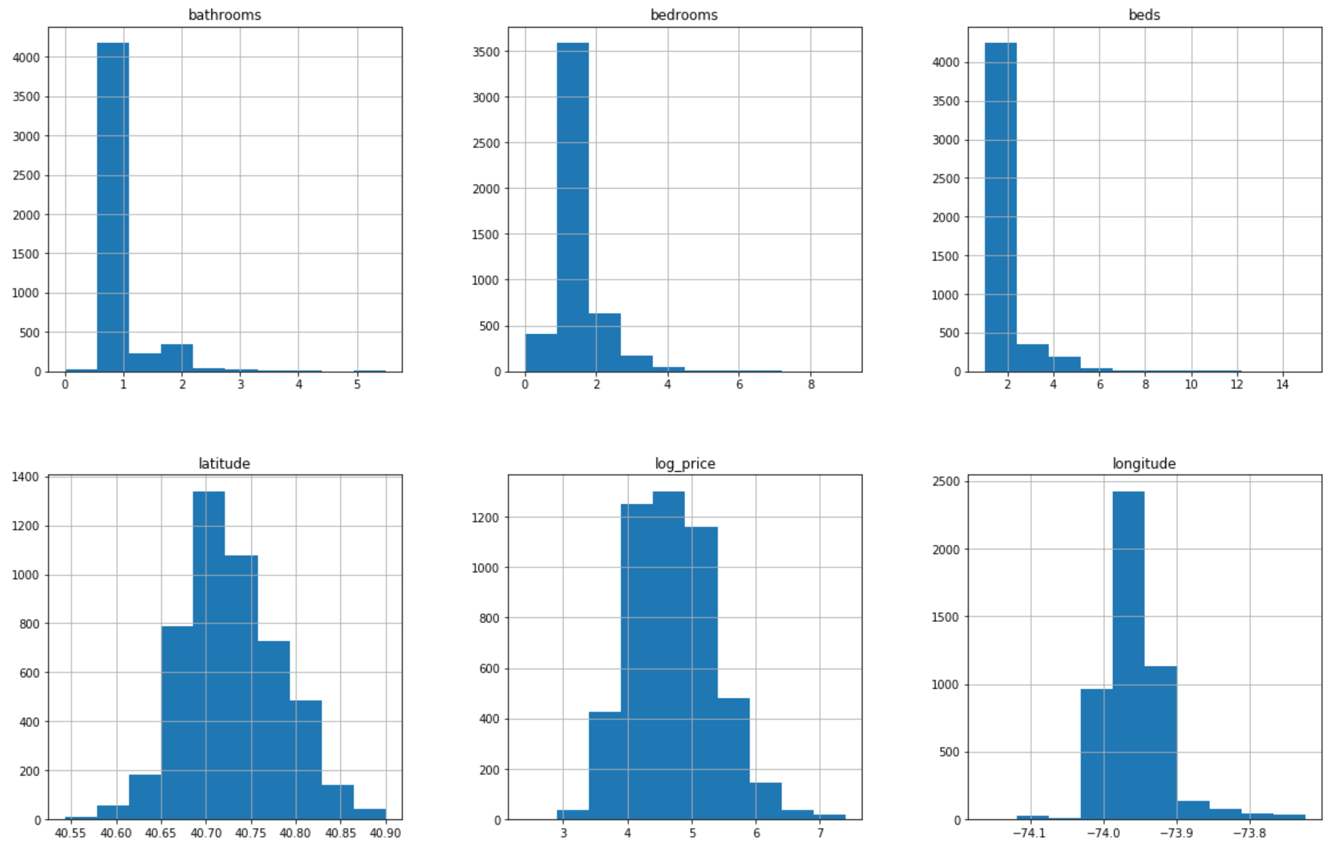


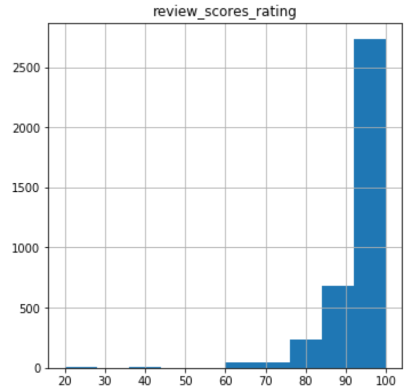
In [14]:

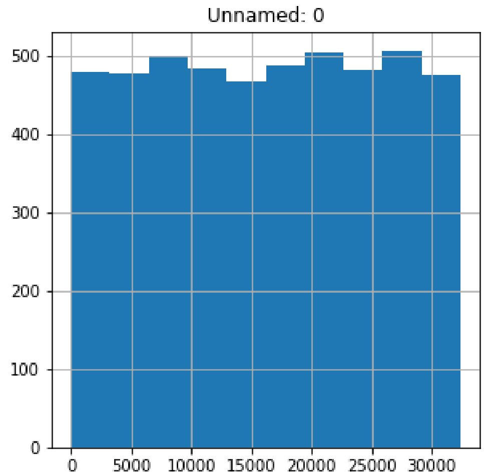
```
my_data[my_data.dtypes[(my_data.dtypes=="float")].index.values].hist(figsize=[20,20])
my_data[my_data.dtypes[(my_data.dtypes=="int64")].index.values].hist(figsize=[11,11])
```

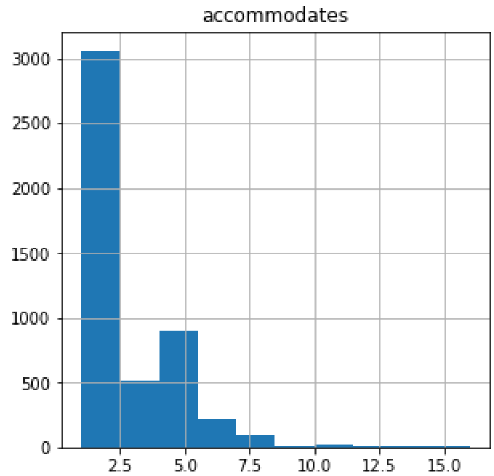
Out[14]:

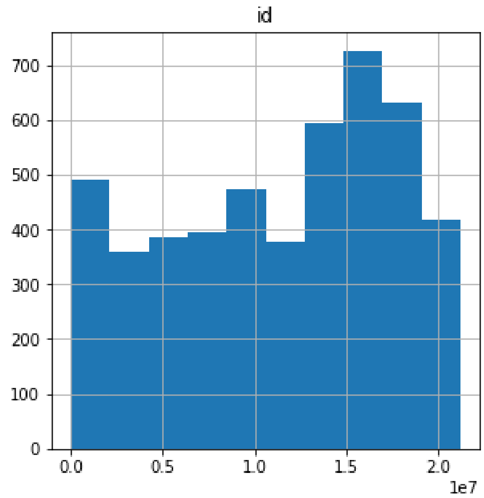
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000294AF9ECCC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000294AFA5C208>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000294AFA94188>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000294AFACD288>]],
      dtype=object)
```

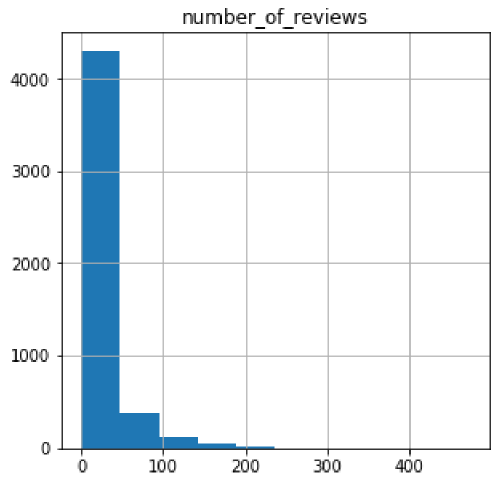












In [15]:

```
my_data.describe().transpose()
```

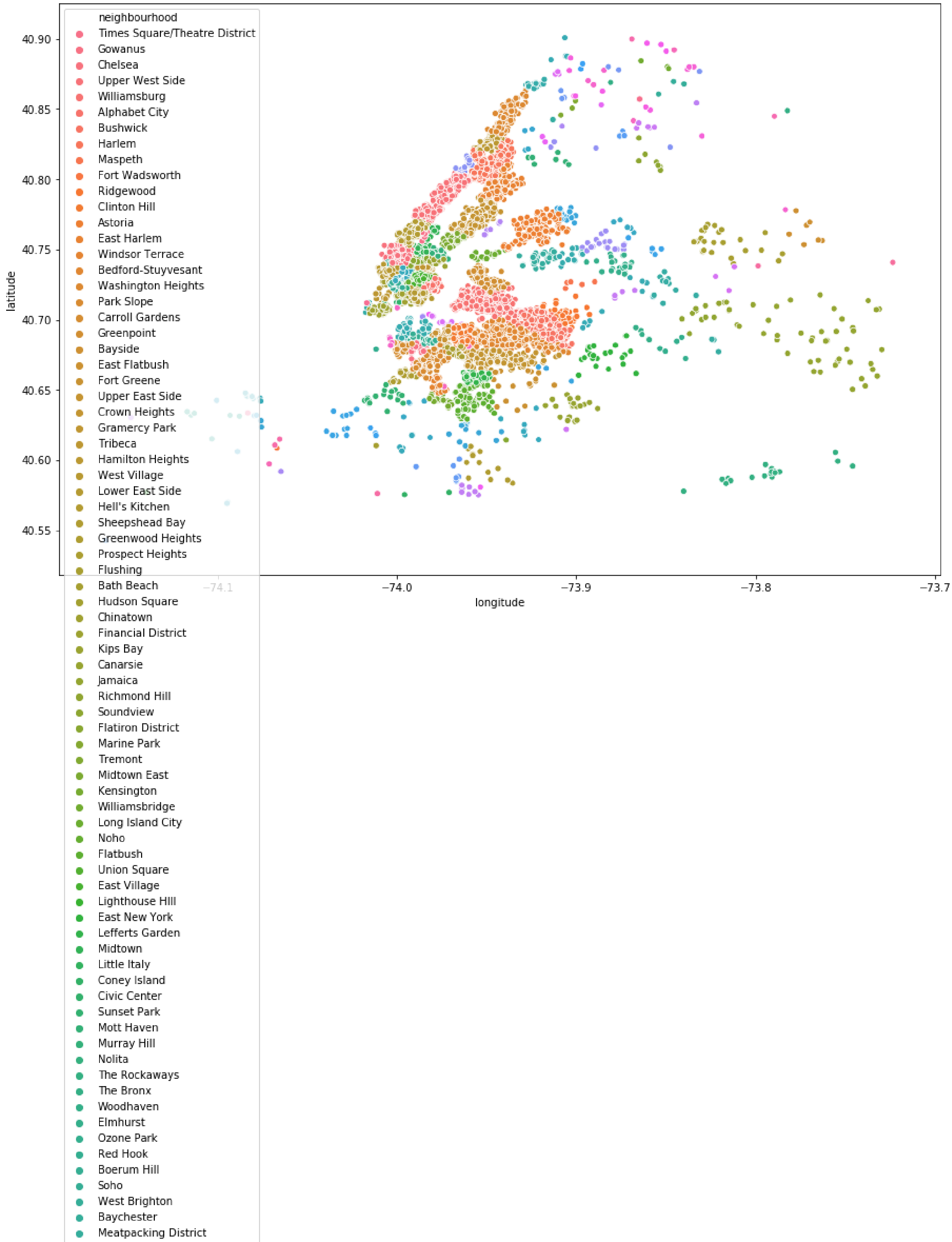
Out[15]:

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	4852.0	1.623937e+04	9.325224e+03	4.000000	8.123500e+03	1.633650e+04	2.442150e+04	3.234800e+04
id	4852.0	1.133776e+07	6.080574e+06	3152.000000	6.213140e+06	1.236683e+07	1.649103e+07	2.117444e+07
log_price	4852.0	4.708381e+00	6.594890e-01	2.397895	4.248495e+00	4.649176e+00	5.164786e+00	7.408531e+00
accommodates	4852.0	2.778648e+00	1.800746e+00	1.000000	2.000000e+00	2.000000e+00	4.000000e+00	1.600000e+01
bathrooms	4838.0	1.122261e+00	3.699294e-01	0.000000	1.000000e+00	1.000000e+00	1.000000e+00	5.500000e+00
latitude	4852.0	4.072880e+01	5.336419e-02	40.542680	4.069014e+01	4.072289e+01	4.076384e+01	4.090080e+01
longitude	4852.0	-7.395477e+01	4.202628e-02	-74.162537	-7.398367e+01	-7.395690e+01	-7.393956e+01	-7.372349e+01
number_of_reviews	4852.0	1.793157e+01	3.270683e+01	0.000000	1.000000e+00	5.000000e+00	2.000000e+01	4.740000e+02

	count	mean	std	min	25%	50%	75%	max
review_scores_rating	3759.0	9.349322e+01	8.581227e+00	20.000000	9.100000e+01	9.600000e+01	1.000000e+02	1.000000e+02
bedrooms	4850.0	1.157938e+00	6.998541e-01	0.000000	1.000000e+00	1.000000e+00	1.000000e+00	9.000000e+00
beds	4840.0	1.537397e+00	1.007436e+00	1.000000	1.000000e+00	1.000000e+00	2.000000e+00	1.500000e+01

In [16]:

```
plt.figure(figsize=(15,10))
sns.scatterplot(my_data.longitude,my_data.latitude,hue=my_data.neighbourhood)
plt.ioff()
```



- Sunnyside
- Mount Eden
- South Ozone Park
- Forest Hills
- Battery Park City
- Cobble Hill
- Inwood
- Downtown Brooklyn
- Woodside
- Brooklyn Heights
- Rego Park
- Riverdale
- Glendale
- St. George
- Bensonhurst
- Greenwich Village
- Flatlands
- Midland Beach
- East Elmhurst
- Borough Park
- Concourse Village
- Randall Manor
- Concord
- Highbridge
- Stapleton
- Midwood
- Ditmars / Steinway
- Brownsville
- Bay Ridge
- Corona
- Dyker Heights
- Kingsbridge Heights
- Eltingville
- Park Versailles
- Gravesend
- University Heights
- Norwood
- Co-op City
- Castle Hill
- Morningside Heights
- Longwood
- Melrose
- Jackson Heights
- DUMBO
- South Beach
- Claremont
- Roosevelt Island
- Elm Park
- Pelham Bay
- Brighton Beach
- Middle Village
- Parkchester
- Bergen Beach
- Kew Garden Hills
- Brooklyn Navy Yard
- Marble Hill
- Belmont
- Concourse
- Manhattan Beach
- Vinegar Hill
- Wakefield
- Fordham
- Morris Park
- Kingsbridge
- Throgs Neck
- Eastchester
- Whitestone
- Bedford Park
- Columbia Street Waterfront
- South Street Seaport
- Van Nest
- Brooklyn
- Edenwald
- Arrochar
- City Island
- Woodlawn
- Rosebank
- Sea Gate
- Utopia
- Queens
- Tompkinsville
- Manhattan
- Bronxdale

In [17]:

```python
#imputing values with mean, median and mode
#mode is 1.0
my_data['bathrooms'].fillna(my_data['bathrooms'].mode()[0],inplace=True)
#mean is 93.50
my_data['review_scores_rating'].fillna(my_data['review_scores_rating'].mean(),inplace=True)
#mode is 1 bedroom
```

```
my_data['bedrooms'].fillna(my_data['bedrooms'].mode()[0],inplace=True)
#mode is 1 bed
my_data['beds'].fillna(my_data['beds'].mode()[0],inplace=True)
```

In [18]:

```
#Checking the number of null count
my_data.isnull().sum()
```

Out[18]:

```
Unnamed: 0                   0
id                           0
log_price                    0
property_type                0
room_type                    0
amenities                    0
accommodates                 0
bathrooms                    0
bed_type                     0
cancellation_policy          0
cleaning_fee                 0
city                         0
description                  0
first_review              1019
host_has_profile_pic        38
host_identity_verified      38
host_response_rate        1504
host_since                  38
instant_bookable             0
last_review               1015
latitude                     0
longitude                    0
name                         0
neighbourhood                0
number_of_reviews            0
review_scores_rating         0
thumbnail_url              358
zipcode                     63
bedrooms                     0
beds                         0
dtype: int64
```

In [19]:

```
#Removing the null values from is_null
Remove_null=pd.DataFrame({"val":my_data['zipcode'].isnull()})
my_data=my_data[Remove_null['val']==False]
```

In [20]:

```
my_data=my_data.drop(['Unnamed: 0','amenities','bed_type','city','description','first_review',

'host_has_profile_pic','host_identity_verified','host_response_rate','host_since'
                       ,'last_review','name','neighbourhood','thumbnail_url','zipcode','id'],axis=1
)
```

In [21]:

```
my_data.isnull().sum()
```

Out[21]:

```
log_price                0
property_type            0
room_type                0
accommodates             0
bathrooms                0
cancellation_policy      0
cleaning_fee             0
instant_bookable         0
latitude                 0
longitude                0
```

```
number_of_reviews       0
review_scores_rating    0
bedrooms                0
beds                    0
dtype: int64
```

In [22]:

```
my_data.describe()
```

Out[22]:

	log_price	accommodates	bathrooms	latitude	longitude	number_of_reviews	review_scores_rating	bedrooms
count	4789.000000	4789.000000	4789.000000	4789.000000	4789.000000	4789.000000	4789.000000	4789.000000
mean	4.705982	2.767592	1.121529	40.728855	-73.954704	17.957194	93.483237	1.156609
std	0.658785	1.785157	0.369942	0.053468	0.042065	32.749212	7.547036	0.697373
min	2.397895	1.000000	0.000000	40.542680	-74.162537	0.000000	20.000000	0.000000
25%	4.248495	2.000000	1.000000	40.690093	-73.983666	1.000000	93.000000	1.000000
50%	4.624973	2.000000	1.000000	40.722996	-73.956807	5.000000	93.493216	1.000000
75%	5.164786	4.000000	1.000000	40.763896	-73.939605	20.000000	98.000000	1.000000
max	7.408531	16.000000	5.500000	40.900803	-73.723488	474.000000	100.000000	9.000000

In [23]:

```
#categorical=['property_type','room_type','cancellation_policy','instant_bookable']
my_data=pd.concat((my_data,pd.get_dummies(my_data['property_type'])),axis=1)
my_data=pd.concat((my_data,pd.get_dummies(my_data['room_type'])),axis=1)
my_data=pd.concat((my_data,pd.get_dummies(my_data['cancellation_policy'])),axis=1)
my_data=pd.concat((my_data,pd.get_dummies(my_data['instant_bookable'])),axis=1)
my_data=pd.concat((my_data,pd.get_dummies(my_data['cleaning_fee'])),axis=1)
```

In [24]:

```
my_data=my_data.drop(['property_type','room_type','cancellation_policy','instant_bookable','cleanin
g_fee'],axis=1)
```

In [25]:

```
my_data.isnull().sum()
```

Out[25]:

```
log_price               0
accommodates            0
bathrooms               0
latitude                0
longitude               0
number_of_reviews       0
review_scores_rating    0
bedrooms                0
beds                    0
Apartment               0
Condo                   0
Hotel type 1            0
Hotel type 2            0
House                   0
Housing                 0
Other                   0
Entire home/apt         0
Private room            0
Shared room             0
flexible                0
moderate                0
strict                  0
Instant Booking         0
No Instant Booking      0
Cleaning Fee Req        0
```

```
No Cleaning Fee          0
dtype: int64
```

In [26]:

```python
target = my_data['log_price']
target_df = pd.DataFrame(target)
target_df.head()
```

Out[26]:

	log_price
0	4.382027
1	5.075174
3	5.010635
4	4.317488
5	5.416100

In [27]:

```python
features_df=my_data.drop(['log_price'],axis=1)
features_df.head()
```

Out[27]:

	accommodates	bathrooms	latitude	longitude	number_of_reviews	review_scores_rating	bedrooms	beds	Apartment	Condo	..
0	1	1.0	40.762239	-73.981589	1	93.493216	1.0	1.0	0	1	..
1	3	1.0	40.677892	-73.992054	17	98.000000	1.0	2.0	1	0	..
3	4	1.0	40.800331	-73.965090	49	96.000000	1.0	2.0	1	0	..
4	2	1.0	40.711386	-73.963529	0	93.493216	1.0	1.0	1	0	..
5	5	1.0	40.726874	-73.979947	0	93.493216	2.0	2.0	1	0	..

5 rows × 25 columns

In [28]:

```python
from sklearn.model_selection import train_test_split
X_train_old, X_test_old, y_train, y_test = train_test_split(features_df,target_df, test_size=0.25,
random_state = 0)
```

In [29]:

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_old)
X_test = scaler.transform(X_test_old)
```

In [30]:

```python
X_train = pd.DataFrame(X_train, columns = X_train_old.columns)
print('Train dataset dimensionality:' , X_train.shape)
print('Train dataset dimensionality:' , y_train.shape)
```

```
Train dataset dimensionality: (3591, 25)
Train dataset dimensionality: (3591, 1)
```

In [31]:

```
X_test = pd.DataFrame(X_test, columns = X_test_old.columns)
print('Test dataset dimensionality:' , X_test.shape)
print('Train dataset dimensionality:' , y_test.shape)
```

```
Test dataset dimensionality: (1198, 25)
Train dataset dimensionality: (1198, 1)
```

Bagging

Bootstrapping - resample method that repetedly drawn sample form smaller data to form smaller data set.

Bagging can be defined as Bootstrapping + Aggrgation and it is an ensemble method in which we first bootstrap our sample data and train them . After that , we aggregate them with equi weights

Model 1 Ridge Regressor with Bagging

In [32]:

```
# Using grid search to find the best parameter for bagging
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
parameters_grid = {'n_estimators': [50,100,200,500],
              'max_samples': [50,100,200,400,500]}

_grid_search_ = GridSearchCV(BaggingRegressor(), parameters_grid, cv=5, return_train_score=True)
_grid_search_.fit(X_train, y_train)
print("Best parameters: {}".format(_grid_search_.best_params_))
print("Best cross-validation score: {:.2f}".format(_grid_search_.best_score_))
```

```
Best parameters: {'max_samples': 500, 'n_estimators': 500}
Best cross-validation score: 0.67
```

In [33]:

```
# Using grid Search to find best parameter for model Ridge Regressor
from  sklearn.linear_model import Ridge
parameters_grid = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
_grid_search_ = GridSearchCV(Ridge(), parameters_grid, cv=5, return_train_score=True)
_grid_search_.fit(X_train, y_train)
print("Best parameters: {}".format(_grid_search_.best_params_))
print("Best cross-validation score: {:.2f}".format(_grid_search_.best_score_))
```

```
Best parameters: {'alpha': 0.1}
Best cross-validation score: 0.58
```

In [34]:

```
from sklearn.ensemble import BaggingRegressor
from  sklearn.linear_model import Ridge

ridge = Ridge(alpha=0.1)
bag_ridge_reg = BaggingRegressor(ridge, n_estimators=500, max_samples=500, bootstrap=True, n_jobs=-
1, random_state=0)

bag_ridge_reg.fit(X_train, y_train)
y_pred = bag_ridge_reg.predict(X_test)

print('Score after applying Bagging on Ridge Regressor on Train data set:
{:.2f}'.format(bag_ridge_reg.score(X_train, y_train)))
print('Score after applying Bagging on Ridge Regressor on Test data set:
```

```
print('Score after applying Bagging on Ridge Regressor on Test data set: {:.2f}'.format(bag_ridge_reg.score(X_test, y_test)))
```

```
Score after applying Bagging on Ridge Regressor on Train data set: 0.59
Score after applying Bagging on Ridge Regressor on Test data set: 0.56
```

Model 2 Decision Tree with Bagging

In [35]:

```
# Grid search to find the best parameter for the model
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
parameters_grid = {'max_depth': np.arange(1, 10)}
_grid_search_ = GridSearchCV(DecisionTreeRegressor(random_state=0), parameters_grid, cv=5, return_train_score=True)
_grid_search_.fit(X_train, y_train)
print("Best parameters: {}".format(_grid_search_.best_params_))
print("Best cross-validation score: {:.2f}".format(_grid_search_.best_score_))
```

```
Best parameters: {'max_depth': 6}
Best cross-validation score: 0.61
```

In [36]:

```
# Grid search to find the best parameter for bagging
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
parameters_grid = {'n_estimators': [50,100,200,500],
              'max_samples': [50,100,200,400,500]}

_grid_search_ = GridSearchCV(BaggingRegressor(), parameters_grid, cv=5, return_train_score=True)
_grid_search_.fit(X_train, y_train)
print("Best parameters: {}".format(_grid_search_.best_params_))
print("Best cross-validation score: {:.2f}".format(_grid_search_.best_score_))
```

```
Best parameters: {'max_samples': 500, 'n_estimators': 500}
Best cross-validation score: 0.67
```

In [37]:

```
# building the bagging model for Decision Tree Regressor using the best parameters
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor

dt_reg = DecisionTreeRegressor(max_depth = 6, random_state=0)
bag_dectree_reg = BaggingRegressor(dt_reg, n_estimators=500, max_samples=500, bootstrap=True, n_jobs=-1, random_state=0)

bag_dectree_reg.fit(X_train, y_train)
y_pred = bag_dectree_reg.predict(X_test)

print('Score after applying Bagging on Decision Tree Regressor on Train data Set: {:.2f}'.format(bag_dectree_reg.score(X_train, y_train)))
print('Score after applying Bagging on Decision Tree Regressor on Test data Set: {:.2f}'.format(bag_dectree_reg.score(X_test, y_test)))
```

```
Score after applying Bagging on Decision Tree Regressor on Train data Set: 0.70
Score after applying Bagging on Decision Tree Regressor on Test data Set: 0.65
```

Pasting

In pasting sampling is done without replacement.Furthermore, bootstrap is set to false in pasting.

Model 1- Lasso Regressor with Pasting

In [38]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
_param_grid = {'n_estimators': [50,100,200,500],
               'max_samples': [50,100,200,400,500]}

__grid__search = GridSearchCV(BaggingRegressor(), _param_grid, cv=5, return_train_score=True)
__grid__search.fit(X_train, y_train)
print("Best parameters: {}".format(__grid__search.best_params_))
print("Best cross-validation score: {:.3f}".format(__grid__search.best_score_))
```

```
Best parameters: {'max_samples': 500, 'n_estimators': 500}
Best cross-validation score: 0.670
```

In [39]:

```
from  sklearn.linear_model import Lasso
lasso_param_ = {'alpha':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
__grid__search = GridSearchCV(Lasso(), lasso_param_, cv=5, return_train_score=True)
__grid__search.fit(X_train, y_train)
print("Best parameters: {}".format(__grid__search.best_params_))
print("Best cross-validation score: {:.3f}".format(__grid__search.best_score_))
```

```
Best parameters: {'alpha': 0.001}
Best cross-validation score: 0.581
```

In [40]:

```
lasso = Lasso(alpha=0.01)
pas_lasso_reg = BaggingRegressor(lasso, n_estimators=500, max_samples=500, bootstrap=False, n_jobs=
-1, random_state=0)

pas_lasso_reg.fit(X_train, y_train)
y_pred = pas_lasso_reg.predict(X_test)

print('Score after aplying Pasting on Lasso Regressor on Train Set: {:.2f}'.format(pas_lasso_reg.sc
ore(X_train, y_train)))
print('Score after pasting Pasting on Lasso Regressor on Test Set: {:.2f}'.format(pas_lasso_reg.sco
re(X_test, y_test)))
```

```
Score after aplying Pasting on Lasso Regressor on Train Set: 0.52
Score after pasting Pasting on Lasso Regressor on Test Set: 0.49
```

Model 2 - Knn Regressor with Pasting

In [41]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
knn_param_grid = {'n_neighbors' : range(1,20), 'p': [1,2], 'weights': ['distance','uniform']}

grid_knn_rgr = GridSearchCV(KNeighborsRegressor(), param_grid = knn_param_grid, cv=10, return_train
_score=True, n_jobs= -1)
grid_knn_rgr.fit(X_train, y_train)

print("Best parameters: {}".format(grid_knn_rgr.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_knn_rgr.best_score_))
```

```
Best parameters: {'n_neighbors': 16, 'p': 1, 'weights': 'distance'}
Best cross-validation score: 0.59
```

In [42]:

```
from sklearn.ensemble import BaggingRegressor
```

```
from sklearn.neighbors import KNeighborsRegressor

bag_reg_knn2 = BaggingRegressor(KNeighborsRegressor(5, p=1, weights= 'distance'),max_features= 9, m
ax_samples=500, n_estimators= 200, random_state=0, bootstrap = False)
bag_reg_knn2.fit(X_train, y_train)
y_pred_knn2 = bag_reg_knn2.predict(X_test)

print('Train score after applying pasting in KNN Regressor:
{:.2f}%'.format(bag_reg_knn2.score(X_train, y_train)*100))
print('Test score after applying pasting in KNN Regresso:
{:.2f}%'.format(bag_reg_knn2.score(X_test, y_test)*100))

print()
print('MAE:', metrics.mean_absolute_error(y_test, y_pred_knn2))
print('MSE:', metrics.mean_squared_error(y_test, y_pred_knn2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_knn2)))
print("r2_Score:",r2_score(y_test, y_pred_knn2))
```

```
Train score after applying pasting in KNN Regressor: 66.03%
Test score after applying pasting in KNN Regresso: 55.84%

MAE: 0.32189785453260594
MSE: 0.19054611432352225
RMSE: 0.43651588095225385
r2_Score: 0.5584223204113332
```

Adaboosting

Adaboost try to fit a sequence of weak learner on reaptedly modified data set

Model 1 - KNN Regressor with Adaboost

In [43]:

```
# Grid search to find the best adaboost
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostRegressor
parameters_grid = {'n_estimators': [50,100,200],
              'learning_rate': [.01, .05, .1, 1]}

__grid_search__ = GridSearchCV(AdaBoostRegressor(random_state = 0), parameters_grid, cv=5, return_t
rain_score=True)
__grid_search__.fit(X_train, y_train)
print("Best parameters: {}".format(__grid_search__.best_params_))
print("Best cross-validation score: {:.2f}".format(__grid_search__.best_score_))
```

```
Best parameters: {'learning_rate': 0.1, 'n_estimators': 50}
Best cross-validation score: 0.58
```

In [44]:

```
from sklearn.neighbors import KNeighborsRegressor
knn_reg = KNeighborsRegressor(n_neighbors=3)
knn_reg.fit(X_train,y_train)
y_pred = knn_reg.predict(X_test)
print('KNN regressor score on Train Set score: {:.2f}'.format(knn_reg.score(X_train, y_train)))
print('KNN regressor score on Test Set score: {:.2f}'.format(knn_reg.score(X_test, y_test)))
```

```
KNN regressor score on Train Set score: 0.77
KNN regressor score on Test Set score: 0.50
```

In [45]:

```
from sklearn.ensemble import AdaBoostRegressor
knn_ada_reg = AdaBoostRegressor(KNeighborsRegressor(n_neighbors=3), n_estimators=100,  learning_rat
```

```
e=0.05, random_state=0)
knn_ada_reg.fit(X_train, y_train)
y_pred = knn_ada_reg.predict(X_test)
print('KNN regressor score on Train Set after applying Adaboost Boosting:
{:.3f}'.format(knn_ada_reg.score(X_train, y_train)))
print('KNN regressor score on Test Set after applying Adaboost Boosting:
{:.3f}'.format(knn_ada_reg.score(X_test, y_test)))
```

```
KNN regressor score on Train Set after applying Adaboost Boosting: 0.869
KNN regressor score on Test Set after applying Adaboost Boosting: 0.488
```

Model 2 - Decision Tree Regressor

In [46]:

```
# Grid search to find the best adaboost
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostRegressor
_parameter_grid = {'n_estimators': [50,100,200,500],
              'learning_rate': [.01, .05, .1, 1]}

grid__search = GridSearchCV(AdaBoostRegressor(random_state = 0), _parameter_grid, cv=5, return_trai
n_score=True)
grid__search.fit(X_train, y_train)
print("Best parameters: {}".format(grid__search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid__search.best_score_))
```

```
Best parameters: {'learning_rate': 0.01, 'n_estimators': 500}
Best cross-validation score: 0.58
```

In [47]:

```
from sklearn.tree import DecisionTreeRegressor
dectr_reg = DecisionTreeRegressor(max_depth = 6, random_state=0)
dectr_reg.fit(X_train, y_train)
y_pred = dectr_reg.predict(X_test)

print('Decision tree regressor score on Train Set score: {:.2f}'.format(dectr_reg.score(X_train, y
_train)))
print('Decision tree regressor score on Test Set score: {:.2f}'.format(dectr_reg.score(X_test,
y_test)))
```

```
Decision tree regressor score on Train Set score: 0.70
Decision tree regressor score on Test Set score: 0.60
```

In [48]:

```
from sklearn.ensemble import AdaBoostRegressor
ada_regr = AdaBoostRegressor(DecisionTreeRegressor(max_depth=6), n_estimators=100,  learning_rate=0
.05, random_state=0)
ada_regr.fit(X_train, y_train)

print('Decision tree regressor score on Train Set after Adaboost Boosting: {:.2f}'.format(ada_regr
.score(X_train, y_train)))
print('Decision tree regressor score on Test Set after Adaboost Boosting: {:.2f}'.format(ada_regr.
score(X_test, y_test)))
```

```
Decision tree regressor score on Train Set after Adaboost Boosting: 0.73
Decision tree regressor score on Test Set after Adaboost Boosting: 0.64
```

Gradient Boosting

In [49]:

```
from  sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
gbrt = GradientBoostingRegressor(random_state=0)
```

```
param_grid={'n_estimators':[50,100,150],'learning_rate':[0.5,1],'max_depth': np.arange(1, 6)}
__grid_search__=GridSearchCV(gbrt,param_grid,cv=5,return_train_score=True)
__grid_search__.fit(X_train, y_train)
print("Best Parameters: {}".format(__grid_search__.best_params_))
print("Best cross-validation score: {:.2f}".format(__grid_search__.best_score_))
```

```
Best Parameters: {'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 50}
Best cross-validation score: 0.66
```

In [50]:

```
# building the model with best parameters
gbrt = GradientBoostingRegressor(random_state=0, learning_rate=0.5, max_depth = 2, n_estimators=50)
gbrt.fit(X_train, y_train)
y_pred = gbrt.predict(X_test)
print("Score after applying Gradient Boosting on Train Set: {:.3f}".format(gbrt.score(X_train,
y_train)))
print("Score after applying  Gradient Boosting on Test Set: {:.3f}".format(gbrt.score(X_test,
y_test)))
```

```
Score after applying Gradient Boosting on Train Set: 0.730
Score after applying  Gradient Boosting on Test Set: 0.655
```

Prinicipal Component Analysis

PCA technique is use to reduce the dimensionality of a data set consisting of many variables correlated to each other.

In [51]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95,random_state = 0)
pca.fit(X_train)
X_train_reduced = pca.transform(X_train)
X_test_reduced = pca.transform(X_test)
```

Checking after PCA how many column get reduced

In [52]:

```
print('X_train shape',X_train.shape)
print('X_train_reduced shape',X_train_reduced.shape)
```

```
X_train shape (3591, 25)
X_train_reduced shape (3591, 8)
```

Scaling the data

In [53]:

```
mm = MinMaxScaler()
X_train_pca= mm.fit_transform(X_train_reduced)
X_test_pca = mm.transform(X_test_reduced)
```

Dummy list will use if required.

In [54]:

```
train_score_pca=[]
test_score_pca=[]
```

```
models_pca =[]
```

Model -1 KNN regressor after PCA Technique

In [55]:

```
np.random.seed(0)

x_range_1 = range(1,30,1)
tuned_parameters=dict(n_neighbors=x_range_1)

#Grid model
knn_reg_pca = KNeighborsRegressor()
grid_knn_pca=GridSearchCV(knn_reg_pca,tuned_parameters,cv=5,return_train_score=True)
grid_model_knn_pca=grid_knn_pca.fit(X_train_pca,y_train)

print(grid_model_knn_pca.best_params_)
print('validation score: {:0.2f}'.format( grid_model_knn_pca.best_score_))
```

```
{'n_neighbors': 7}
validation score: 0.53
```

In [56]:

```
#General model
knn_pca=KNeighborsRegressor(n_neighbors=9)
knn_model_pca=knn_pca.fit(X_train_pca,y_train)
print('Train score: {}'.format(knn_model_pca.score(X_train_pca,y_train)))
print('Test score: {}'.format(knn_model_pca.score(X_test_pca,y_test)))
train_score_pca.append(knn_model_pca.score(X_train_pca,y_train))
test_score_pca.append(knn_model_pca.score(X_test_pca,y_test))
```

```
Train score: 0.6375807251169314
Test score: 0.5246936528801134
```

In [57]:

```
#calculating the accuracies
knn_accuracies_pca = cross_val_score(estimator = knn_model_pca, X = X_train_pca, y = y_train, cv =
10)
print("Accuracy: {:.2f} %".format(knn_accuracies_pca.mean()*100))
```

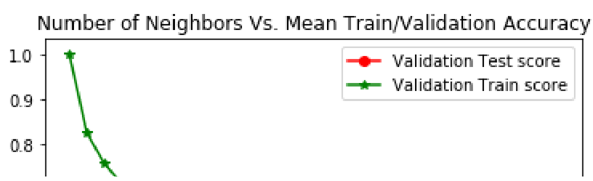
```
Accuracy: 53.17 %
```

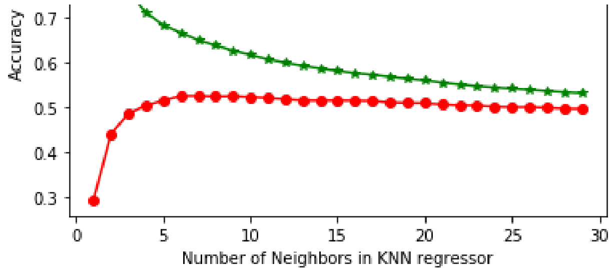
In [58]:

```
#visualizing the train and test accuracy score for KNN Regressor
import matplotlib.pyplot as plt
views = pd.DataFrame(grid_knn_pca.cv_results_)
plt.plot(views['param_n_neighbors'],views['mean_test_score'],marker='o',c='r',label='Validation
Test score')
plt.plot(views['param_n_neighbors'],views['mean_train_score'],marker='*',c='g',label='Validation Tr
ain score')
plt.title('Number of Neighbors Vs. Mean Train/Validation Accuracy')
plt.xlabel('Number of Neighbors in KNN regressor')
plt.ylabel('Accuracy')
plt.legend()
```

Out[58]:

```
<matplotlib.legend.Legend at 0x294b1c71188>
```





In [59]:

```
#train and test accuracy score for KNN Regressor after running PCA
knn = KNeighborsRegressor(n_neighbors=7)
knn.fit(X_train_pca, y_train)
print('Train score on best parameters for KNN Regressor
{}'.format(knn.score(X_train_pca,y_train)))
print('Test score on best parameters for KNN Regressor {}'.format(knn.score(X_test_pca,y_test)))
```

```
Train score on best parameters for KNN Regressor 0.6570193928545962
Test score on best parameters for KNN Regressor 0.5201252133003174
```

Model 2 -Linear Regression after PCA

In [60]:

```
#train and test accuracy score for linear regression after running PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
lreg = LinearRegression()
#scaled train test
lreg.fit(X_train_pca, y_train)
print('Training score for regression model: {}'.format(lreg.score(X_train_pca, y_train)))
print('Testing score for regression model: {}'.format(lreg.score(X_test_pca, y_test)))
print('R-squared score (training): {:.2f}\n'.format(lreg.score(X_train_pca, y_train)))
print('R-squared score (test): {:.2f}'.format(lreg.score(X_test_pca, y_test)))
train_score_pca.append(lreg.score(X_train_pca, y_train))
test_score_pca.append(lreg.score(X_test_pca, y_test))
```

```
Training score for regression model: 0.4594824287389822
Testing score for regression model: 0.43657755698958867
R-squared score (training): 0.46

R-squared score (test): 0.44
```

In [61]:

```
#calculating the accuracies
lnr_accuracies_pca = cross_val_score(estimator = lreg, X = X_train_pca, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(lnr_accuracies_pca.mean()*100))
```

```
Accuracy: 45.56 %
```

Model 3 - Ridge regressor after PCA

In [62]:

```
np.random.seed(0)
x_range_2 = [0.01, 0.1, 1, 10, 100]
tuned_parameters = [{'alpha':x_range_2}]

#Grid model
ridge_pca = Ridge(max_iter=1000,tol=0.1,random_state=0)
grid_ridge_pca=GridSearchCV(ridge_pca,tuned_parameters,cv=5, return_train_score= True, iid = False)
grid_model_ridge_pca=grid_ridge_pca.fit(X_train_pca,y_train)

print("Best parameters: {}".format(grid_model_ridge_pca.best_params_))
```

```
print('Best parameters: {}'.format(grid_model_ridge_pca.best_params_))
print('Cross validation score: {:0.2f}'.format( grid_model_ridge_pca.best_score_))
```

```
Best parameters: {'alpha': 1}
Cross validation score: 0.45
```

In [63]:

```
#General model
ridge_1_pca=Ridge(alpha=0.1)
ridge_model_pca=ridge_1_pca.fit(X_train_pca,y_train)
print('Training score for Ridge regression model:
{}'.format(ridge_model_pca.score(X_train_pca,y_train)))
print('Testing score for Ridge regression model:
{}'.format(ridge_model_pca.score(X_test_pca,y_test)))
train_score_pca.append(ridge_model_pca.score(X_train_pca,y_train))
test_score_pca.append(ridge_model_pca.score(X_test_pca,y_test))
```

```
Training score for Ridge regression model: 0.459482267198l1064
Testing score for Ridge regression model: 0.436596l6180333627
```

In [64]:

```
#calculating the accuracies
ridr_accuracies_pca = cross_val_score(estimator = ridge_model_pca, X = X_train_pca, y = y_train, cv
= 10)
print("Accuracy: {:.2f} %".format(ridr_accuracies_pca.mean()*100))
```

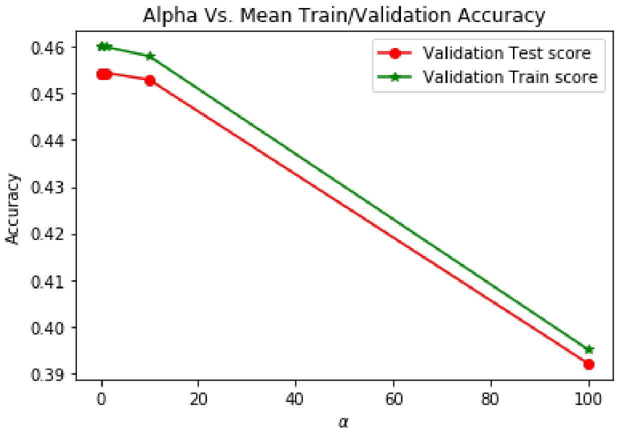
```
Accuracy: 45.56 %
```

In [65]:

```
#visualizing the train and test accuracy score for Ridge regression
import matplotlib.pyplot as plt
vector = pd.DataFrame(grid_model_ridge_pca.cv_results_)
plt.plot(vector['param_alpha'],vector['mean_test_score'],marker='o',c='r',label='Validation Test sc
ore')
plt.plot(vector['param_alpha'],vector['mean_train_score'],marker='*',c='g',label='Validation Train
score')
plt.title('Alpha Vs. Mean Train/Validation Accuracy')
plt.xlabel(r'$\alpha$')
plt.ylabel('Accuracy')
plt.legend()
```

Out[65]:

```
<matplotlib.legend.Legend at 0x294b1ce8908>
```



Model 4 Lasso Regression after PCA

In [66]:

```
#list the best parameter value for lasso regressor
from  sklearn.linear_model import Lasso
np.random.seed(0)
x_range_3 = [0.01, 0.1, 1, 10, 100]
tuned_parameters = [{'alpha':x_range_3}]

#Grid model
lasso_pca = Lasso(max_iter=1000,tol=0.1,random_state=0)
grid_lasso_pca=GridSearchCV(lasso_pca,tuned_parameters,cv=5, return_train_score= True, iid = False)
grid_model_lasso_pca=grid_lasso_pca.fit(X_train_pca,y_train)

print("Best parameters: {}".format(grid_model_lasso_pca.best_params_))
print('Best Crossvalidation score: {:0.2f}'.format( grid_model_lasso_pca.best_score_))
```

```
Best parameters: {'alpha': 0.01}
Best Crossvalidation score: 0.42
```

In [67]:

```
#General model
lasso_1_pca=Lasso(alpha=0.01, tol=0.1)
lasso_model_pca=lasso_1_pca.fit(X_train_pca,y_train)
print('Train score on best parameters for Lasso Regressor:
{}'.format(lasso_model_pca.score(X_train_pca,y_train)))
print('Test score on best parameters for Lasso Regressor:
{}'.format(lasso_model_pca.score(X_test_pca,y_test)))
train_score_pca.append(lasso_model_pca.score(X_train_pca,y_train))
test_score_pca.append(lasso_model_pca.score(X_test_pca,y_test))
```

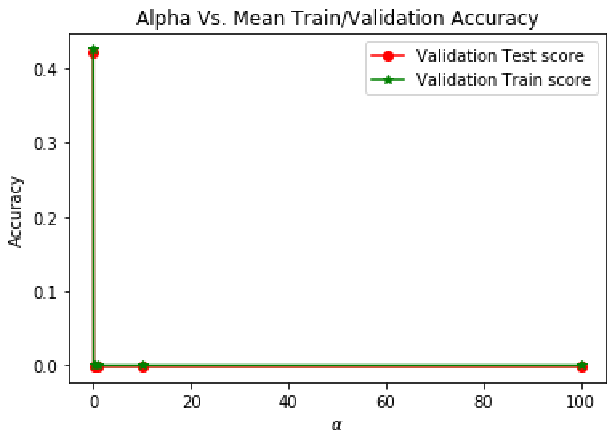
```
Train score on best parameters for Lasso Regressor: 0.42537317346395176
Test score on best parameters for Lasso Regressor: 0.40669138836971286
```

In [68]:

```
#visualizing the train and test accuracy score for lasso regression
import matplotlib.pyplot as plt
vis_results = pd.DataFrame(grid_lasso_pca.cv_results_)
plt.plot(vis_results['param_alpha'],vis_results['mean_test_score'],marker='o',c='r',label='Validati
on Test score')
plt.plot(vis_results['param_alpha'],vis_results['mean_train_score'],marker='*',c='g',label='Validat
ion Train score')
plt.title('Alpha Vs. Mean Train/Validation Accuracy')
plt.xlabel(r'$\alpha$')
plt.ylabel('Accuracy')
plt.legend()
```

Out[68]:

```
<matplotlib.legend.Legend at 0x294b1d89f48>
```



Model 5 -Polynomial Regerssion after PCA

In [69]:

```
from  sklearn.preprocessing  import PolynomialFeatures
train_score_list = []
test_score_list = []
regressor = LinearRegression()
for n in range(1,4):
    poly = PolynomialFeatures(n)
    X_train_poly_pca = poly.fit_transform(X_train_pca)
    X_test_poly_pca = poly.transform(X_test_pca)
    regressor.fit(X_train_poly_pca, y_train)
    train_score_list.append(regressor.score(X_train_poly_pca, y_train))
    test_score_list.append(regressor.score(X_test_poly_pca, y_test))

train = [sum(train_score_list)/len(train_score_list)]
test = [sum(test_score_list)/len(test_score_list)]
print(train)
print(test)
```

```
[0.5509409687032404]
[0.5029318211353652]
```

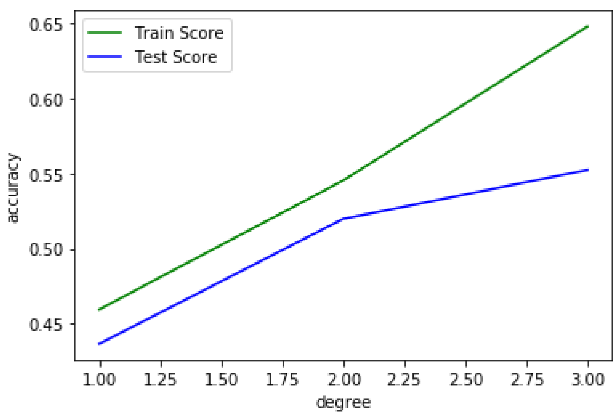
Plotting Accuracy graph for Polynomial Regression after

In [70]:

```
x_axis = range(1,4)
plt.plot(x_axis, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()
```

Out[70]:

```
<matplotlib.legend.Legend at 0x294b1d89a08>
```



Model 5 - Support Vector Machine after PCA

In [71]:

```
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
_paramtr_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}

grid_search = GridSearchCV(SVR(), _paramtr_grid, cv=5, return_train_score=True)
grid_search.fit(X_train_pca, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'C': 10, 'gamma': 100}
Best cross-validation score: 0.55
```

Linear SVM after PCA

In [72]:

```
ln_svr = LinearSVR(C=10).fit(X_train_pca, y_train)
print('Train score on best parameters for LinearSVR -
{}'.format(ln_svr.score(X_train_pca,y_train)))
print('Test score on best parameters for LinearSVR - {}'.format(ln_svr.score(X_test_pca,y_test)))
```

```
Train score on best parameters for LinearSVR - 0.4565816063539583
Test score on best parameters for LinearSVR - 0.43280524491002903
```

In [73]:

```
#calculating the accuracy
ln_svr_accuracy = cross_val_score(estimator = ln_svr, X = X_train_pca, y = y_train, cv = 5)
print("Accuracy: {:.2f} %".format(ln_svr_accuracy.mean()*100))
```

```
Accuracy: 45.17 %
```

Kernel(Linear)SVM after PCA

In [74]:

```
kl_svm = SVR(kernel='linear', C=10).fit(X_train_pca, y_train)
print('Train score on best parameters for SVR kernel - Linear {}'.format(kl_svm.score(X_train_pca,
y_train)))
print('Test score on best parameters for SVR kernel - Linear {}'.format(kl_svm.score(X_test_pca,y_
test)))
```

```
Train score on best parameters for SVR kernel - Linear 0.45574929544318354
Test score on best parameters for SVR kernel - Linear 0.42599964884893393
```

In [75]:

```
#calculating the accuracy
kl_svm_accuracy = cross_val_score(estimator = kl_svm, X = X_train_pca, y = y_train, cv = 5)
print("Accuracy: {:.2f} %".format(kl_svm_accuracy.mean()*100))
```

```
Accuracy: 45.20 %
```

Kernel(RBF) SVM after PCA

In [76]:

```
krbf_svm = SVR(kernel='rbf', gamma=100, C=10).fit(X_train_pca, y_train)
print('Train score on best parameters for SVR kernel - rbf {}'.format(krbf_svm.score(X_train_pca,y
_train)))
print('Test score on best parameters for SVR kernel - rbf {}'.format(krbf_svm.score(X_test_pca,y_t
est)))
```

```
Train score on best parameters for SVR kernel - rbf 0.6782168760741613
Test score on best parameters for SVR kernel - rbf 0.5401128539537601
```

In [77]:

```
#calculating the accuracy
krbf_svm_accuracy = cross_val_score(estimator = krbf_svm, X = X_train_pca, y = y_train, cv = 5)
print("Accuracy: {:.2f} %".format(krbf_svm_accuracy.mean()*100))
```

```
Accuracy: 55.19 %
```

Kernel SVM (Poly) after PCA

In [78]:

```
klp_svvm = SVR(kernel='poly', degree=3, C=10).fit(X_train_pca, y_train)
print('Train score on best parameters for SVR kernel - poly {}'.format(klp_svvm.score(X_train_pca,
y_train)))
print('Test score on best parameters for SVR kernel - poly {}'.format(klp_svvm.score(X_test_pca,y_
test)))
```

```
Train score on best parameters for SVR kernel - poly 0.4498346524343754
Test score on best parameters for SVR kernel - poly 0.42378354017324316
```

In [79]:

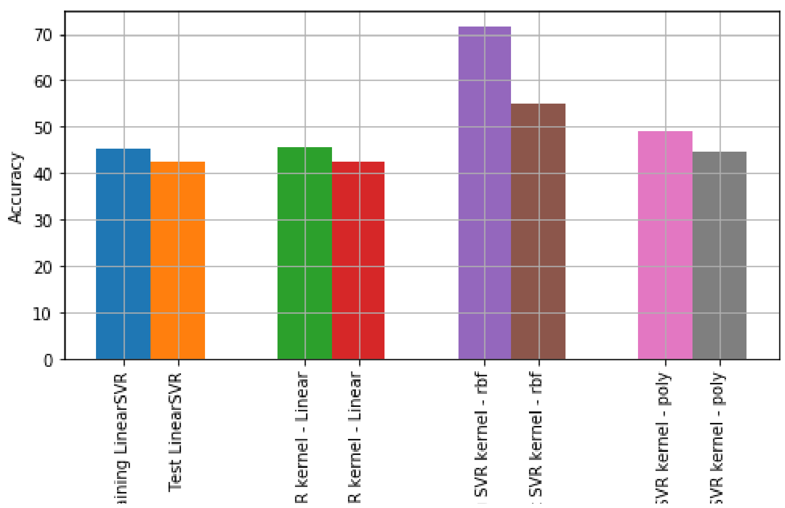
```
#calculating the accuracy of Kernal SVM(Poly)
klp_svvm_accuracy = cross_val_score(estimator = klp_svvm, X = X_train_pca, y = y_train, cv = 5)
print("Accuracy: {:.2f} %".format(klp_svvm_accuracy.mean()*100))
```

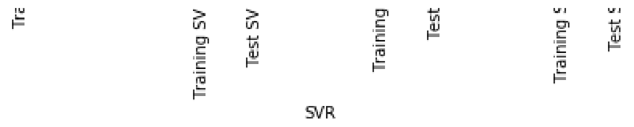
```
Accuracy: 43.41 %
```

Visualizating the train and test score of kernalized SVM regression models after applying PCA

In [80]:

```
fig, ax = plt.subplots(figsize=(8,4))
width = 0.3
plt.xlabel('SVR')
plt.ylabel('Accuracy')
outer_labels = ['Training LinearSVR','Test LinearSVR','Training SVR kernel - Linear','Test SVR ker
nel - Linear','Training SVR kernel - rbf',
          'Test SVR kernel - rbf','Training SVR kernel - poly','Test SVR kernel - poly']
inner_label = ['Training LinearSVR','Test LinearSVR','Training SVR kernel - Linear','Test SVR kern
el - Linear','Training SVR kernel - rbf',
          'Test SVR kernel - rbf','Training SVR kernel - poly','Test SVR kernel - poly']
list_1 = [0,.3,1,1.3,2,2.3,3,3.3]
ax.set_xticks(list_1)
for j in range(0,4,1) :
    ax.set_xticklabels(outer_labels,rotation=90)
    ax.set_xticklabels(inner_label,rotation=90)
train_accuracylist=[45.38,45.57,71.57,49.05]
test_accuracylist=[42.42,42.59,55.01,44.76]
for i in range(0,4,1) :
    ax.bar(i,train_accuracylist[i],width)
    ax.bar(i+width,test_accuracylist[i],width)
plt.grid()
```





Compairing Model before applying PCA technique and after applying PCA technique

In [81]:

```
#Index Levels
outer_row= ['Before PCA','Before PCA','After PCA','After PCA']
inner_row = ['Training Accuracy','Test Accuracy','Training Accuracy','Test Accuracy']
grn_level = list(zip(outer_row,inner_row))
grn_level = pd.MultiIndex.from_tuples(grn_level)
```

In [82]:

```
#train and test accuracy score of the models as observed before and after PCA
data_subset = np.array([(0.5935,0.5900,0.6514,0.5935,0.5894,0.3634,0.8042,0.5825,0.5612),
                        (0.5582,0.5581,0.5782,0.5585,0.5567,0.3875,0.5161,0.5425,0.5212),
                        (0.4594,0.5509,0.6375,0.4594,0.4253,0.4481,0.6782,0.4557,0.4498),
                        (0.4365,0.5029,0.5246,0.4365,0.4066,0.4144,0.5401,0.4259,0.4237)])
```

In [83]:

```
data_df = pd.DataFrame(data_subset,index=grn_level,columns=['Linear Regression','Polynomial
Regressor','KNN Regressor',
                                                          'Ridge Regressor','Lasso Regressor','Linear
R','SVM - RBF Kernel','SVM - Linear Kernel','SVM - Poly Kernel'])
```

Comparison Table for our Model

In [84]:

```
import seaborn as sns

cm = sns.light_palette("Purple", as_cmap=True)

s = data_df.style.background_gradient(cmap='tab20b')
s
```

Out[84]:

		Linear Regression	Polynomial Regressor	KNN Regressor	Ridge Regressor	Lasso Regressor	Linear SVR	SVM - RBF Kernel	SVM - Linear Kernel	SVM - Poly Kernel
Before PCA	Training Accuracy	0.5935	0.59	0.6514	0.5935	0.5894	0.3634	0.8042	0.5825	0.5612
	Test Accuracy	0.5582	0.5581	0.5782	0.5585	0.5567	0.3875	0.5161	0.5425	0.5212
After PCA	Training Accuracy	0.4594	0.5509	0.6375	0.4594	0.4253	0.4481	0.6782	0.4557	0.4498
	Test Accuracy	0.4365	0.5029	0.5246	0.4365	0.4066	0.4144	0.5401	0.4259	0.4237

Observations

It is observed from the above steps PCA reduce the dimenstionality from 25 to 8 . Primary reason for using PCA is that it decrease time complexity of our model.Furthermore,Computation time decrease but as a penalty accuracy of the model decreases.If we refer out comparison table above before PCA model accuracy and after PCA accuracy we can find out that majority of our

model performed well before PCA and few of the model accuracy are well in after PCA.Therefore we will go ahead with before PCA model though it has higher dimensionality but accuracy is better.

Deep Learning Model - Regression: Neural Networks

In [86]:

```
#install the packages
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

```
Using TensorFlow backend.
```

In [87]:

```
'''
Steps  for creating Neural Network using Keras Classifier

'''

# Step 1:Create model
model = Sequential()
# Defining Input layer
model.add(Dense(24, input_dim = 25, activation = 'relu'))
# Defining Hidden layer
model.add(Dense(50, activation = 'relu'))
model.add(Dense(25, activation = 'relu'))
#Output layer
model.add(Dense(1, kernel_initializer='normal',activation = 'sigmoid'))

# Step 2: Build the computational graph - compile
model.compile(loss = 'mean_absolute_error', optimizer = 'adam', metrics = ['mean_absolute_error'] )

# Step 3: Train the model
model.fit(X_train, y_train, epochs = 30, batch_size = 50)
```

```
Epoch 1/30
3591/3591 [==============================] - 1s 207us/step - loss: 3.9610 - mean_absolute_error: 3
.9610
Epoch 2/30
3591/3591 [==============================] - 0s 70us/step - loss: 3.6922 - mean_absolute_error: 3.
6922
Epoch 3/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6915 - mean_absolute_error: 3.
6915
Epoch 4/30
3591/3591 [==============================] - 0s 70us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 5/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 6/30
3591/3591 [==============================] - 0s 78us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 7/30
3591/3591 [==============================] - 0s 78us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 8/30
3591/3591 [==============================] - 0s 79us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 9/30
3591/3591 [==============================] - 0s 81us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 10/30
3591/3591 [==============================] - 0s 76us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 11/30
```

```
3591/3591 [==============================] - 0s 67us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 12/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 13/30
3591/3591 [==============================] - 0s 70us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 14/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 15/30
3591/3591 [==============================] - 0s 67us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 16/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 17/30
3591/3591 [==============================] - 0s 70us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 18/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 19/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 20/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 21/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 22/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 23/30
3591/3591 [==============================] - 0s 79us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 24/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 25/30
3591/3591 [==============================] - 0s 68us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 26/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 27/30
3591/3591 [==============================] - 0s 71us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 28/30
3591/3591 [==============================] - 0s 70us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 29/30
3591/3591 [==============================] - 0s 77us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
Epoch 30/30
3591/3591 [==============================] - 0s 69us/step - loss: 3.6914 - mean_absolute_error: 3.
6914
```

Out[87]:

```
<keras.callbacks.callbacks.History at 0x294c3901288>
```

In [88]:

```python
# Summarize Result
loss_and_metrics = model.evaluate(X_test, y_test)

print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

```
1198/1198 [==============================] - 0s 67us/step
Test Loss 3.749836247433008
Test Accuracy 3.7498362064361572
```

In [ ]:

```

```

Classification- Bank Marketing

Bank Marketing csv The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Information: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

There are 21 columns and 41188 rows in this dataset. We have imported all the necessary files and libraries. We also filled the null values with mean and did the visualization using seaborn, pyplot, matplotlib.

Variables - age, job, marital, education, default, housing, loan, contact, month, day_of_week, duration, campaign, pdays , previous, poutcome, emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed, y.

We will perform Classification on this dataset.

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Read csv file

In [2]:

```
bnk = pd.read_csv(r"C:\Users\SST190000\Downloads\Applied ML\archive\bank_marketing.csv", sep = ';')
```

In [3]:

```
bnk
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre

41188 rows × 21 columns

Getting information about the dataset

In [4]:

```
bnk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [5]:

```
bnk.describe()
```

Out[5]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.00000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.62129
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.73444
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.63400
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.34400
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.85700
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.96100
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.04500

In [6]:

```
bnk.columns
```

Out[6]:

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

In [7]:

```
bnk.shape
```

Out[7]:

```
(41188, 21)
```

Getting information about the dataset

In [8]:

```
bnk.isnull().sum()
```

Out[8]:

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

Check for Duplicates and Null values

In [9]:

```
bnk.duplicated().sum()
```

Out[9]:

```
12
```

In [10]:

```
duplicates = bnk[bnk.duplicated()]
print("Duplicate rows : ")
duplicates
```

```
Duplicate rows :
```

Out[10]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre
1266	39	blue-collar	married	basic.6y	no	no	no	telephone	may	thu	...	1	999	
12261	36	retired	married	unknown	no	no	no	telephone	jul	thu	...	1	999	
14234	27	technician	single	professional.course	no	no	no	cellular	jul	mon	...	2	999	
16956	47	technician	divorced	high.school	no	yes	no	cellular	jul	thu	...	3	999	

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	pre
16956	47	technician	divorced	high.school	no	yes	no	cellular	jul	thu	...	3	999	
18465	32	technician	single	professional.course	no	yes	no	cellular	jul	thu	...	1	999	
20216	55	services	married	high.school	unknown	no	no	cellular	aug	mon	...	1	999	
20534	41	technician	married	professional.course	no	yes	no	cellular	aug	tue	...	1	999	
25217	39	admin.	married	university.degree	no	no	no	cellular	nov	tue	...	2	999	
28477	24	services	single	high.school	no	yes	no	cellular	apr	tue	...	1	999	
32516	35	admin.	married	university.degree	no	yes	no	cellular	may	fri	...	4	999	
36951	45	admin.	married	university.degree	no	no	no	cellular	jul	thu	...	1	999	
38281	71	retired	single	university.degree	no	no	no	telephone	oct	tue	...	1	999	

12 rows × 21 columns

Drop duplicates

In [11]:

```
bnk.drop_duplicates(keep=False, inplace=True)
bnk.duplicated().sum()
```

Out[11]:

```
0
```

In [12]:

```
bnk.isna().sum()
```

Out[12]:

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

Imputing 5%-10% null values

In [13]:

```
bnk['emp.var.rate'] = bnk['emp.var.rate'].mask(np.random.random(bnk['emp.var.rate'].shape) < .1)
bnk['duration'] = bnk['duration'].mask(np.random.random(bnk['duration'].shape) < .05)
bnk['campaign'] = bnk['campaign'].mask(np.random.random(bnk['campaign'].shape) < .1)
bnk['pdays'] = bnk['pdays'].mask(np.random.random(bnk['pdays'].shape) < .1)
bnk['previous'] = bnk['previous'].mask(np.random.random(bnk['previous'].shape) < .05)
```

In [14]:

```
bnk.isna().sum()
```

Out[14]:

```
age                  0
job                  0
marital              0
education            0
default              0
housing              0
loan                 0
contact              0
month                0
day_of_week          0
duration          2138
campaign          4085
pdays             4088
previous          1980
poutcome             0
emp.var.rate      4192
cons.price.idx       0
cons.conf.idx        0
euribor3m            0
nr.employed          0
y                    0
dtype: int64
```

In [15]:

```
bnk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41164 non-null  int64  
 1   job             41164 non-null  object 
 2   marital         41164 non-null  object 
 3   education       41164 non-null  object 
 4   default         41164 non-null  object 
 5   housing         41164 non-null  object 
 6   loan            41164 non-null  object 
 7   contact         41164 non-null  object 
 8   month           41164 non-null  object 
 9   day_of_week     41164 non-null  object 
 10  duration        39026 non-null  float64
 11  campaign        37079 non-null  float64
 12  pdays           37076 non-null  float64
 13  previous        39184 non-null  float64
 14  poutcome        41164 non-null  object 
 15  emp.var.rate    36972 non-null  float64
 16  cons.price.idx  41164 non-null  float64
 17  cons.conf.idx   41164 non-null  float64
 18  euribor3m       41164 non-null  float64
 19  nr.employed     41164 non-null  float64
 20  y               41164 non-null  object 
dtypes: float64(9), int64(1), object(11)
memory usage: 6.9+ MB
```

Replace null values with mean

In [16]:

```
bnk['emp.var.rate'].fillna(bnk['emp.var.rate'].mean(),inplace=True)
bnk['duration'].fillna(bnk['duration'].mean(),inplace=True)
bnk['campaign'].fillna(bnk['campaign'].mean(),inplace=True)
bnk['pdays'].fillna(bnk['pdays'].mean(),inplace=True)
bnk['previous'].fillna(bnk['previous'].mean(),inplace=True)
```

In [17]:

```
bnk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41164 non-null  int64  
 1   job             41164 non-null  object 
 2   marital         41164 non-null  object 
 3   education       41164 non-null  object 
 4   default         41164 non-null  object 
 5   housing         41164 non-null  object 
 6   loan            41164 non-null  object 
 7   contact         41164 non-null  object 
 8   month           41164 non-null  object 
 9   day_of_week     41164 non-null  object 
 10  duration        41164 non-null  float64
 11  campaign        41164 non-null  float64
 12  pdays           41164 non-null  float64
 13  previous        41164 non-null  float64
 14  poutcome        41164 non-null  object 
 15  emp.var.rate    41164 non-null  float64
 16  cons.price.idx  41164 non-null  float64
 17  cons.conf.idx   41164 non-null  float64
 18  euribor3m       41164 non-null  float64
 19  nr.employed     41164 non-null  float64
 20  y               41164 non-null  object 
dtypes: float64(9), int64(1), object(11)
memory usage: 6.9+ MB
```

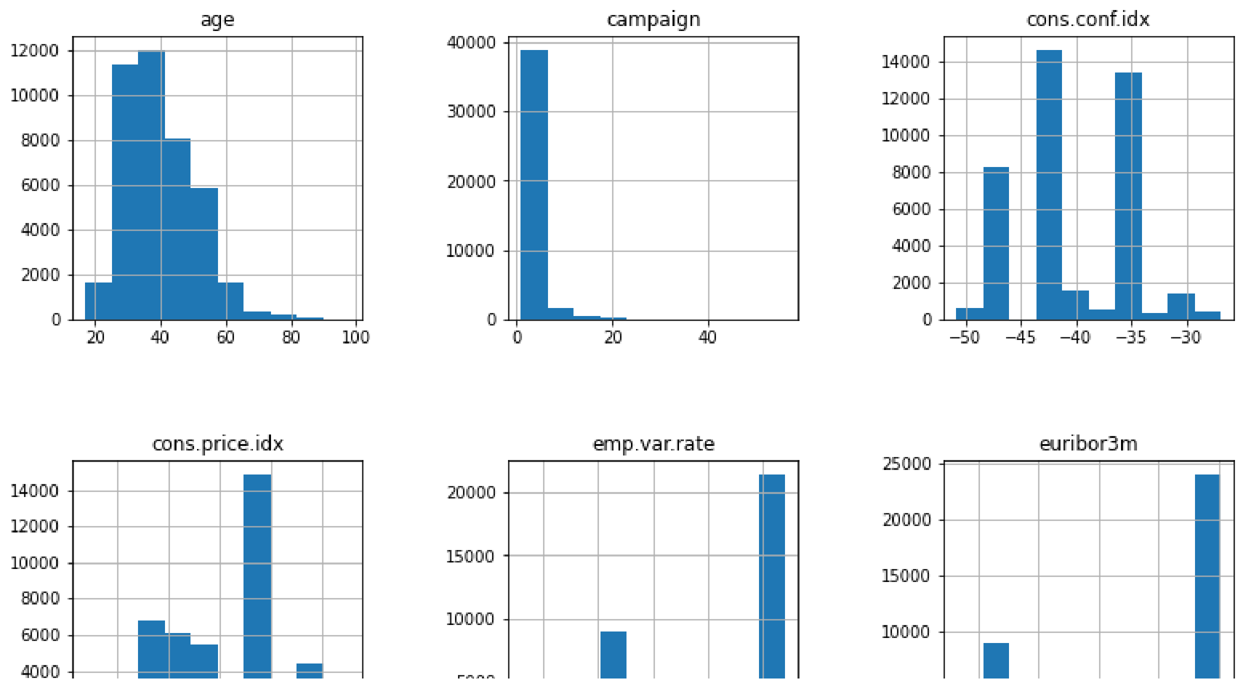
Exploratory Data Analysis

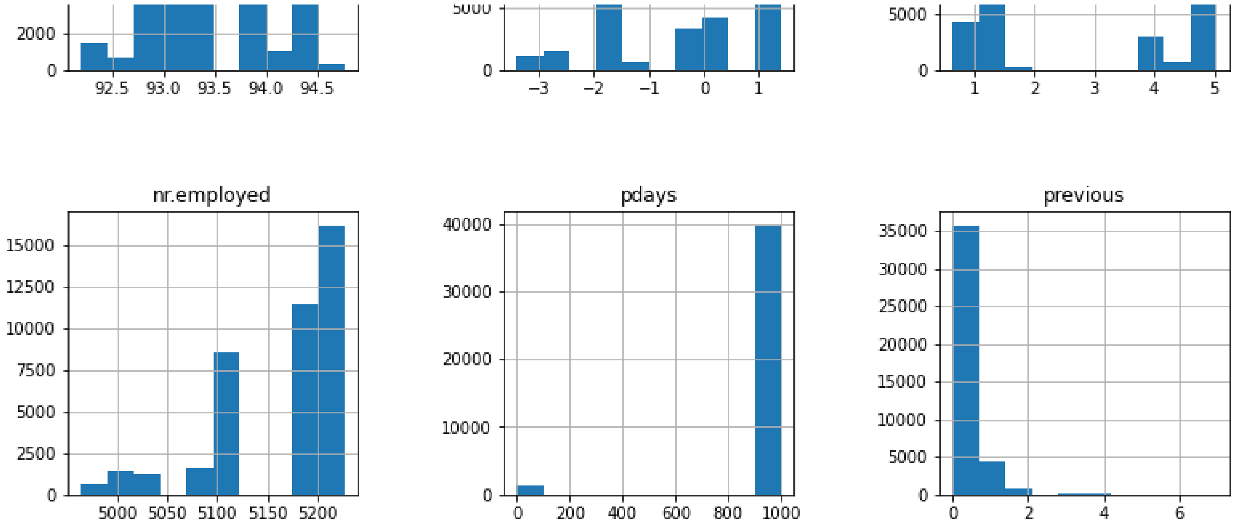
Exploring numerical variables in 'bnk'

Histogram Subplots

In [18]:

```
col = ['age','campaign','pdays','previous','emp.var.rate','cons.price.idx','cons.conf.idx','euribor3m','nr.employed']
bnk.hist(column=col,figsize=(13,13))
plt.subplots_adjust(wspace = 0.5, hspace = 0.5)
plt.show()
```





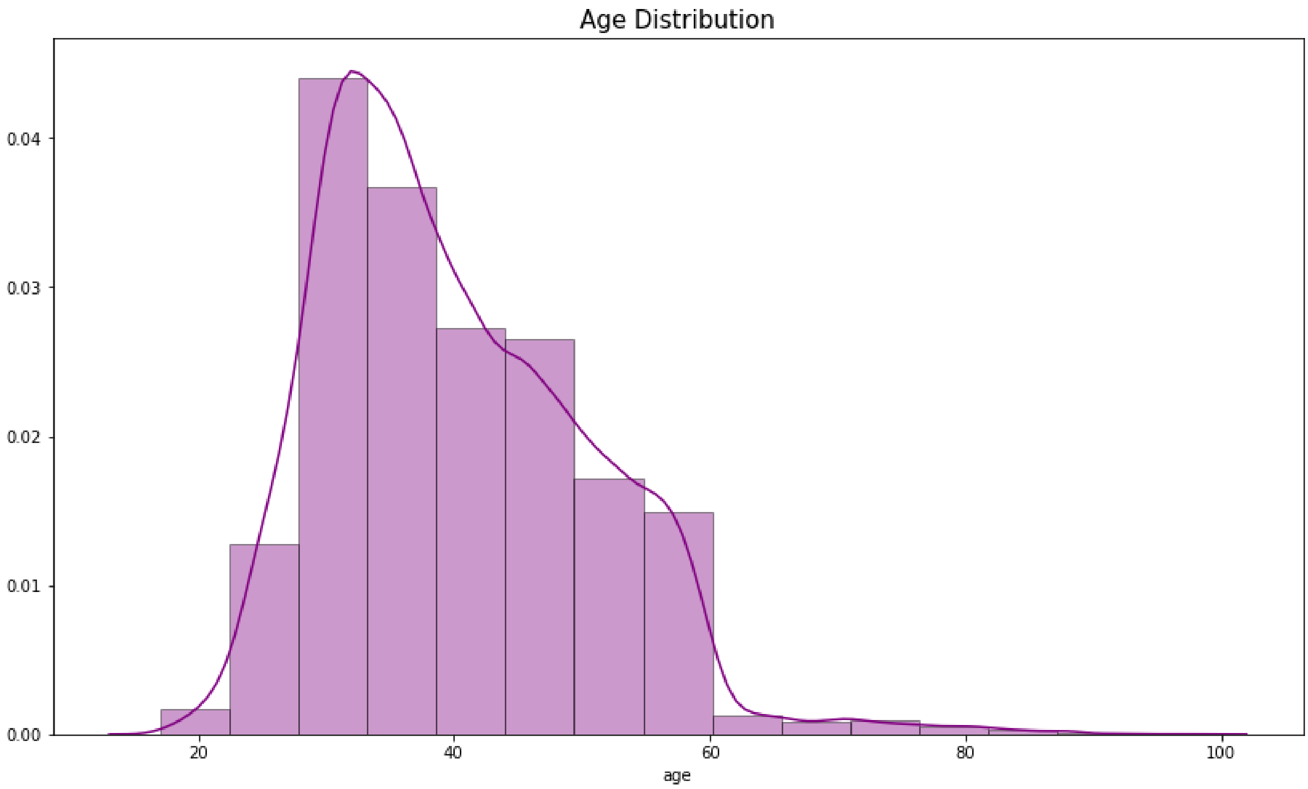
Distribution of age variable

In [19]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
sns.distplot(bnk['age'], hist=True, kde=True,
             bins=int(150/10), color = 'purple',
             hist_kws={'edgecolor':'black'})
bca.set_title('Age Distribution', fontsize=15)
```

Out[19]:

```
Text(0.5, 1.0, 'Age Distribution')
```



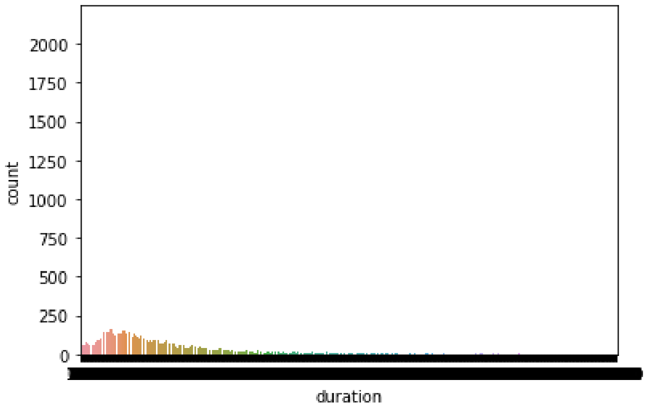
Count of Duration

In [20]:

```
sns.countplot(x='duration',data=bnk)
bca.set_title('Count of Duration', fontsize=15)
```

Out[20]:

```
Text(0.5, 1.0, 'Count of Duration')
```



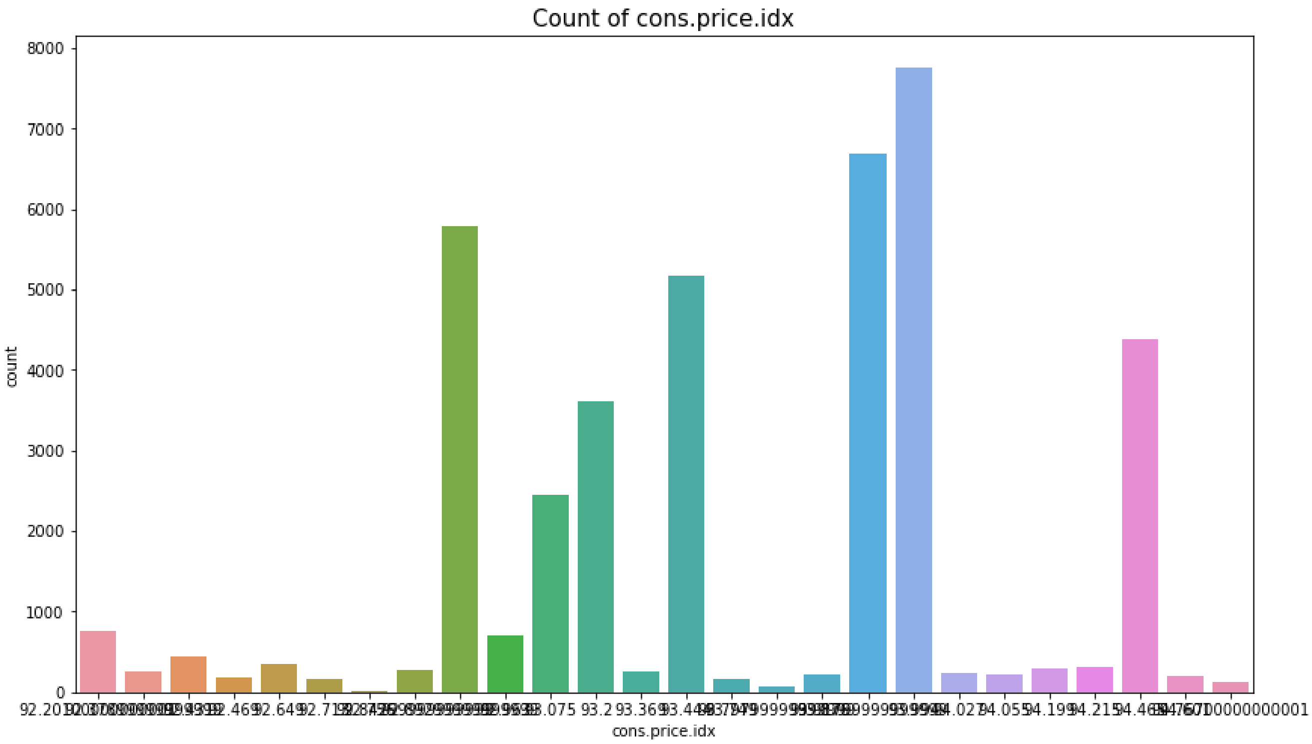
Count of cons.price.idx

In [21]:

```python
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
sns.countplot(x='cons.price.idx',data=bnk)
bca.set_title('Count of cons.price.idx', fontsize=15)
```

Out[21]:

```
Text(0.5, 1.0, 'Count of cons.price.idx')
```



Count of emp.var.rate

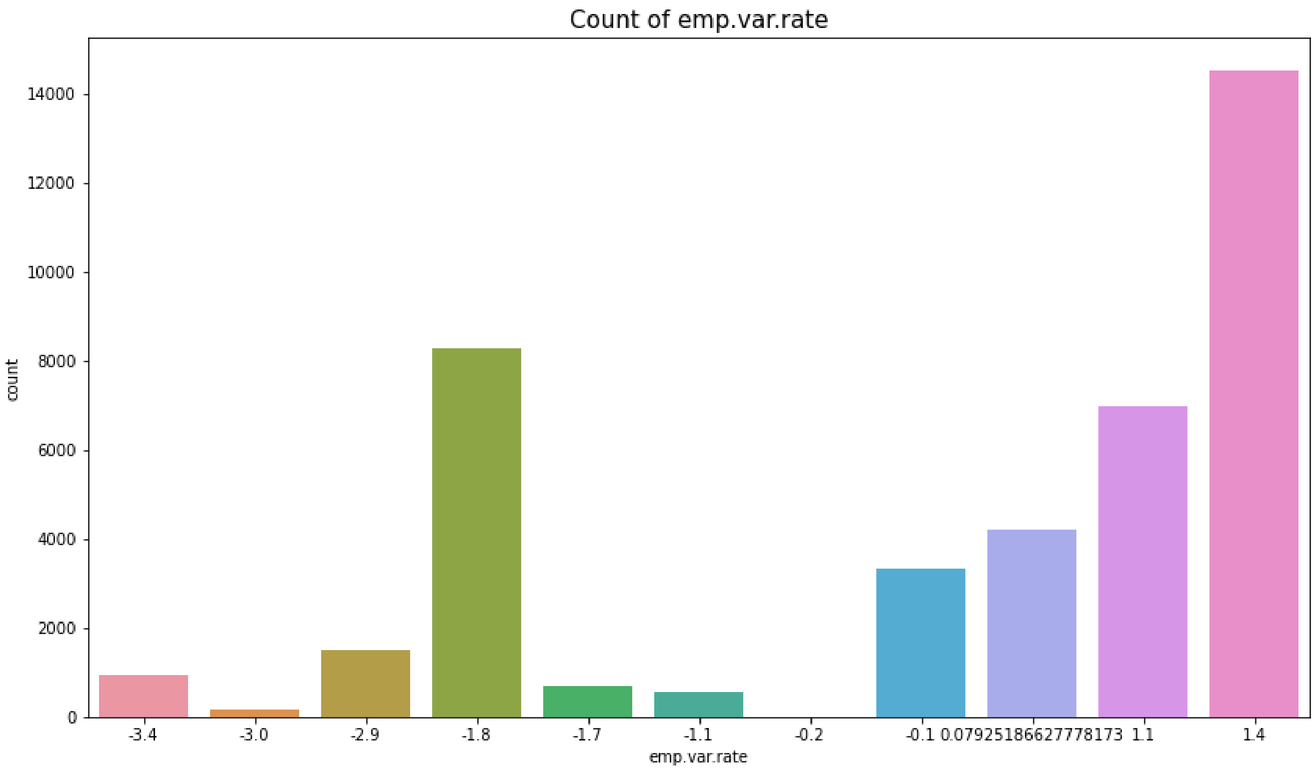
In [22]:

```python
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
sns.countplot(x='emp.var.rate',data=bnk)
bca.set_title('Count of emp.var.rate', fontsize=15)
```

```
bca.set_title('Count of emp.var.rate', fontsize=15)
```

Out[22]:

```
Text(0.5, 1.0, 'Count of emp.var.rate')
```



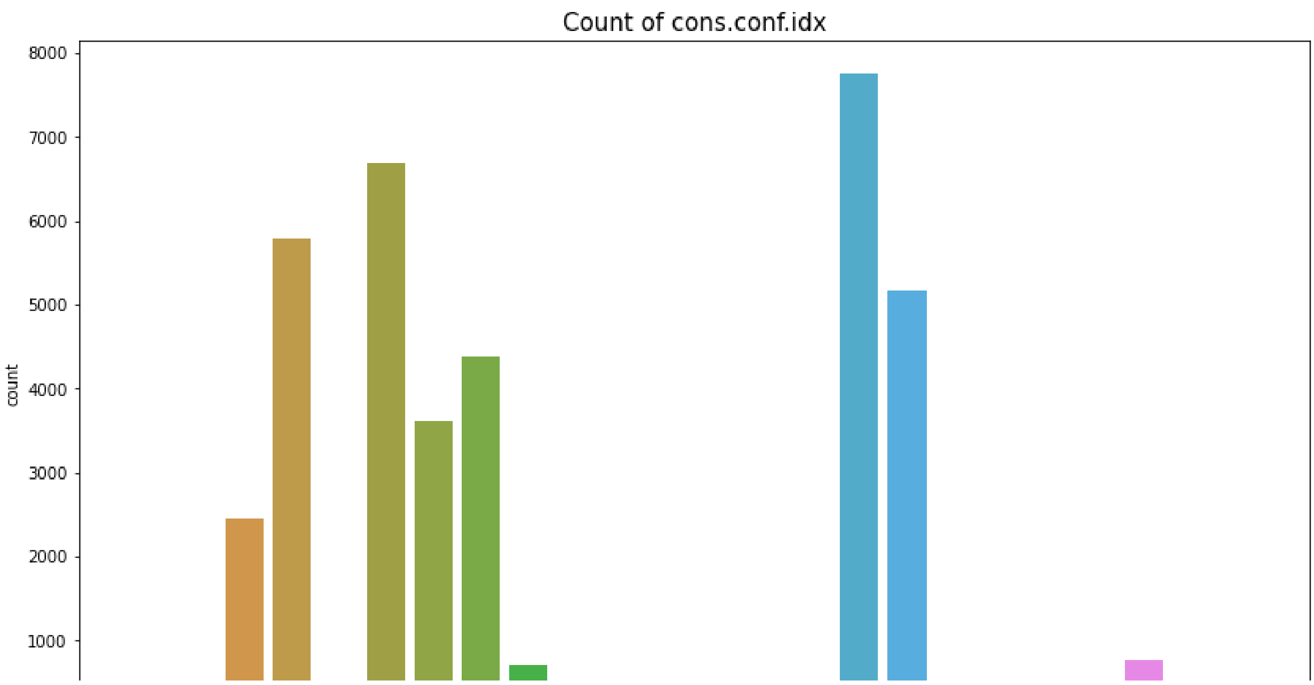
Count of cons.conf.idx

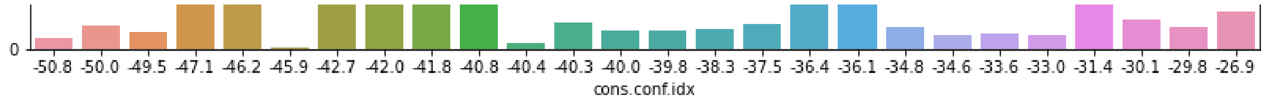
In [23]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
sns.countplot(x='cons.conf.idx',data=bnk)
bca.set_title('Count of cons.conf.idx', fontsize=15)
```

Out[23]:

```
Text(0.5, 1.0, 'Count of cons.conf.idx')
```





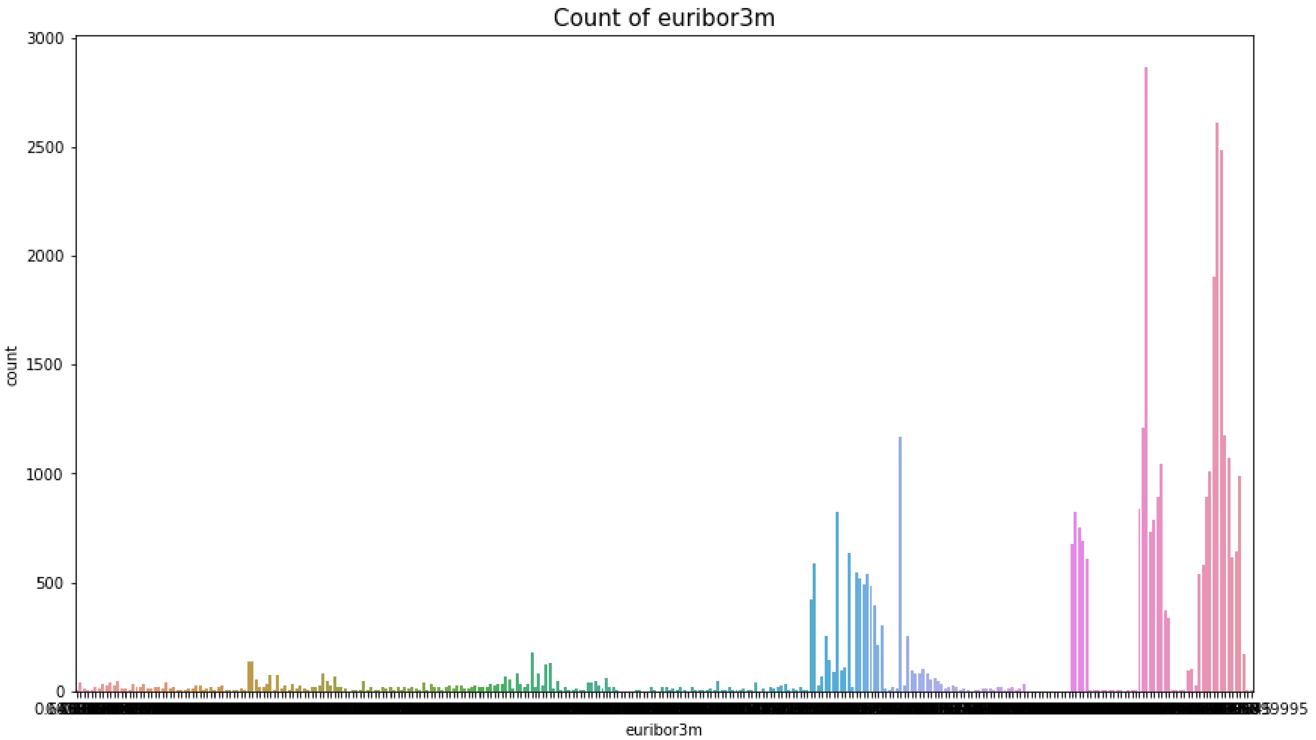
Count of euribor3m

In [24]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
sns.countplot(x='euribor3m',data=bnk)
bca.set_title('Count of euribor3m', fontsize=15)
```

Out[24]:

```
Text(0.5, 1.0, 'Count of euribor3m')
```

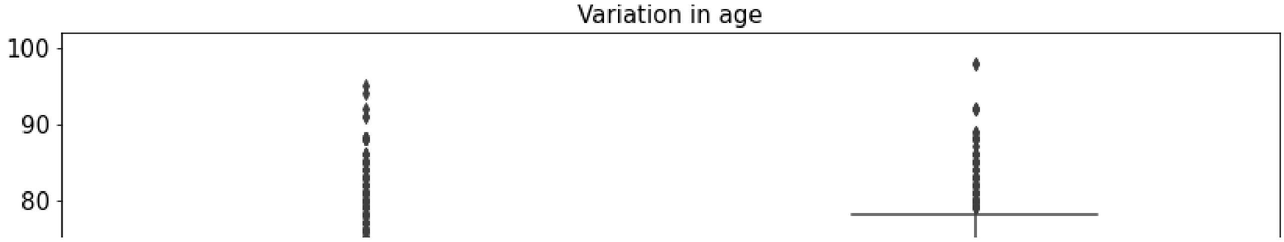


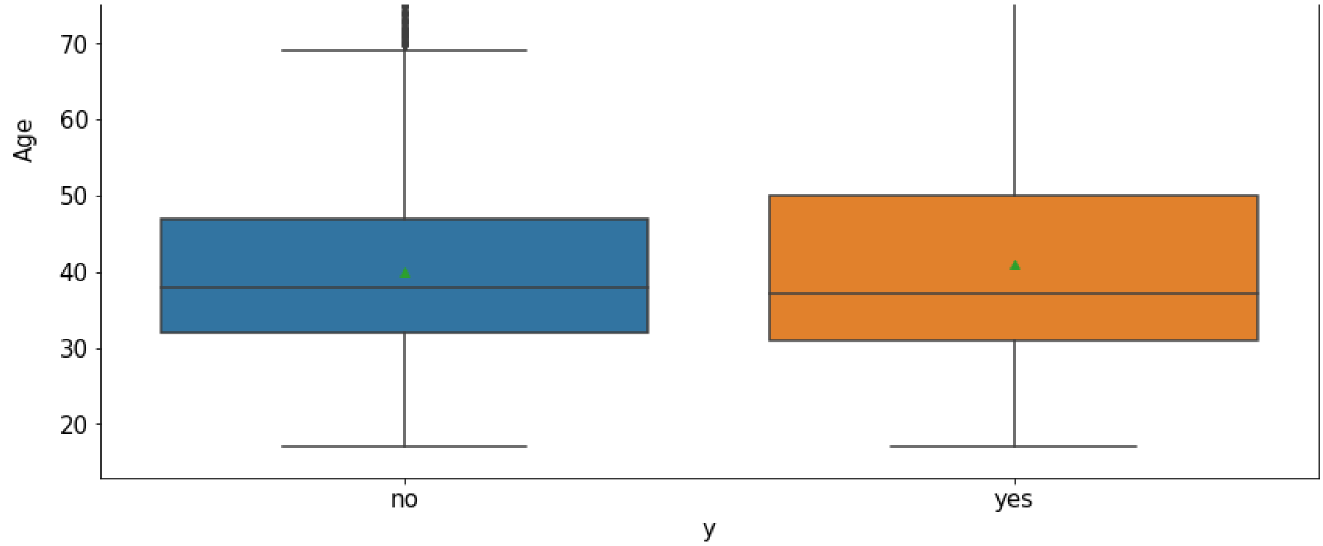
Exploring variation of numerical variables w.r.t target variable y

Variation in age

In [25]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'age', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Age', fontsize=15)
bca1.set_title('Variation in age', fontsize=15)
bca1.tick_params(labelsize=15)
```

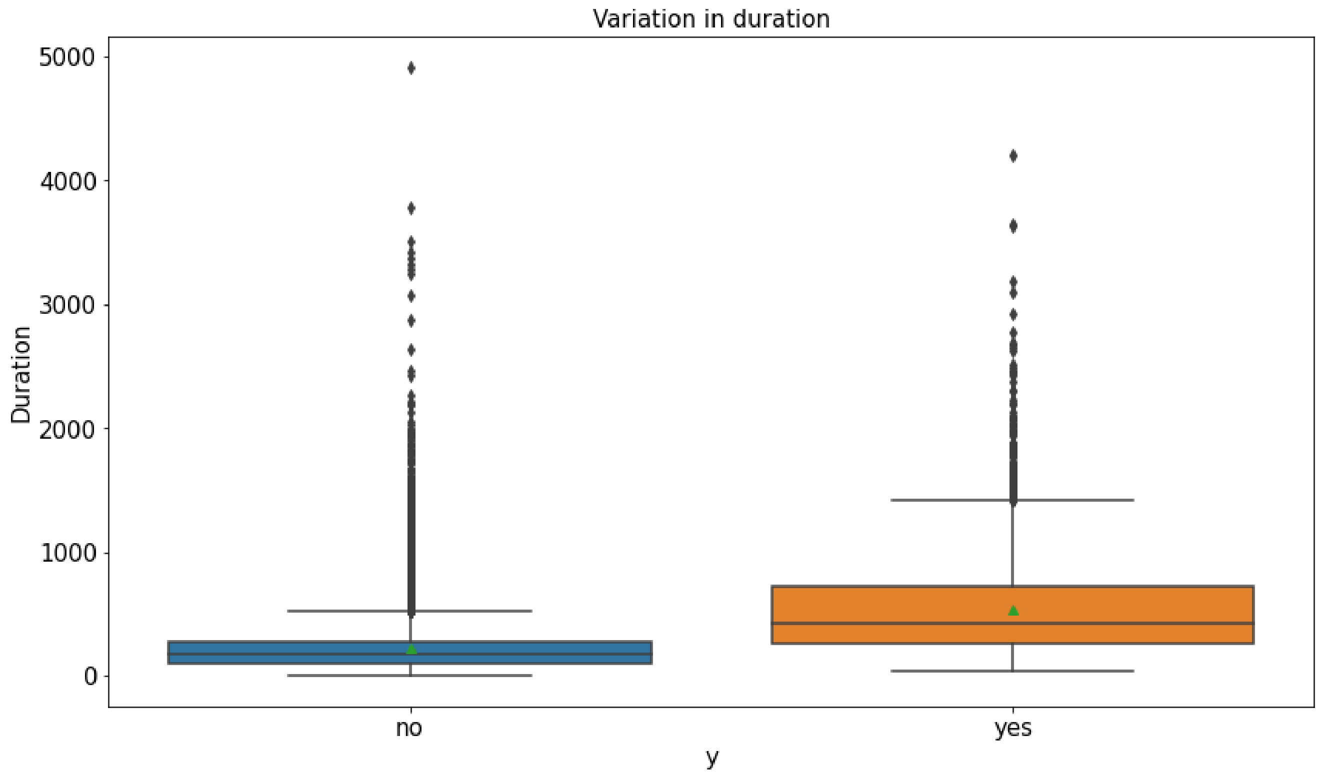




Variation in duration

In [26]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'duration', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Duration', fontsize=15)
bca1.set_title('Variation in duration', fontsize=15)
bca1.tick_params(labelsize=15)
```

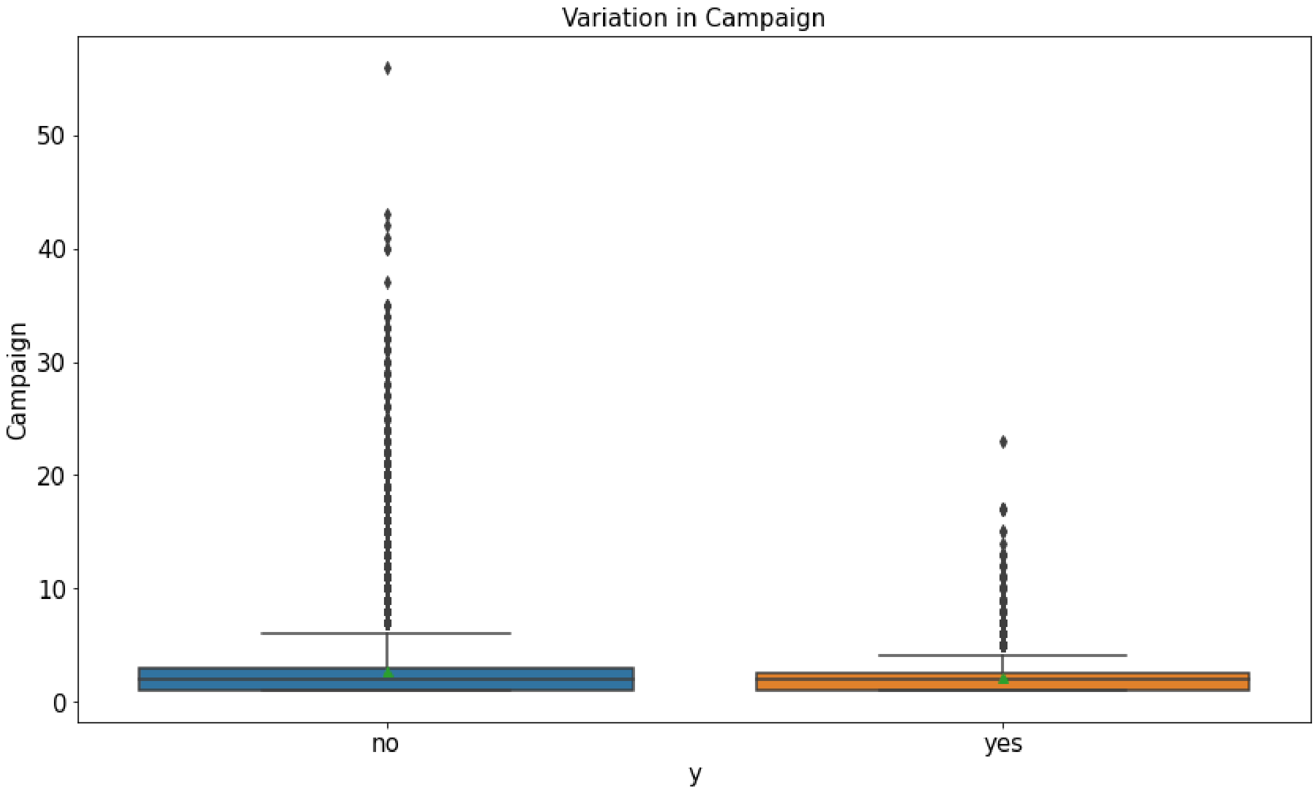


Variation in Campaign

In [27]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'campaign', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Campaign', fontsize=15)
```

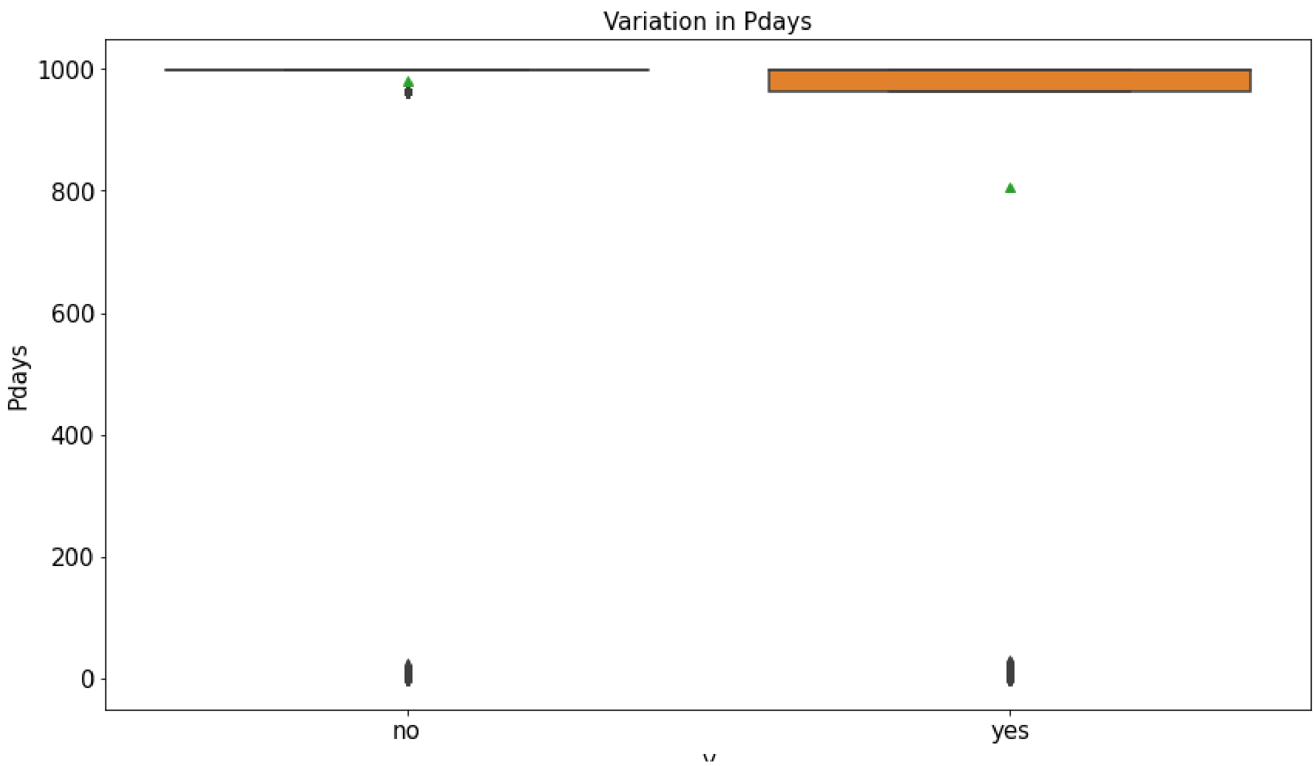
```
bca1.set_title('Variation in Campaign', fontsize=15)
bca1.tick_params(labelsize=15)
```



Variation in Pdays

In [28]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'pdays', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Pdays', fontsize=15)
bca1.set_title('Variation in Pdays', fontsize=15)
bca1.tick_params(labelsize=15)
```

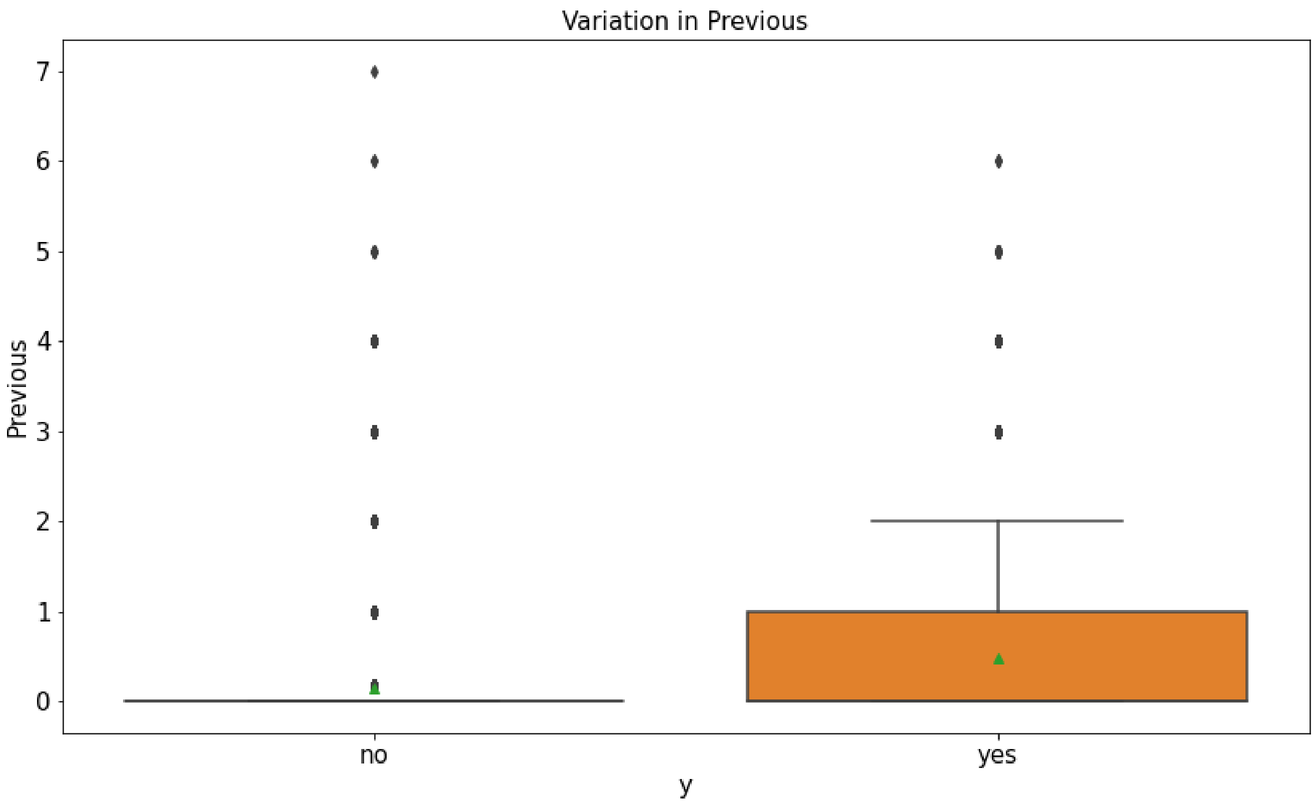


y

Variation in Previous

In [29]:

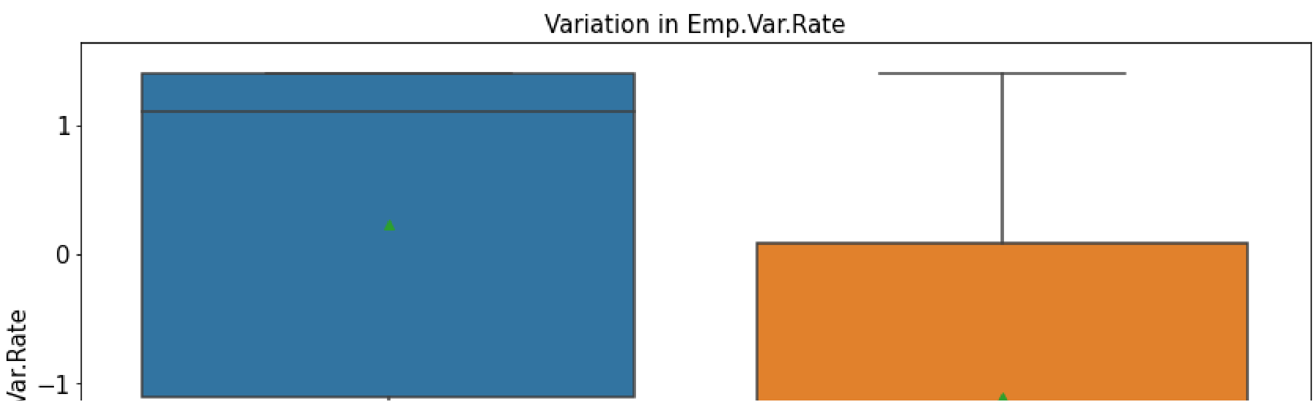
```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'previous', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Previous', fontsize=15)
bca1.set_title('Variation in Previous', fontsize=15)
bca1.tick_params(labelsize=15)
```

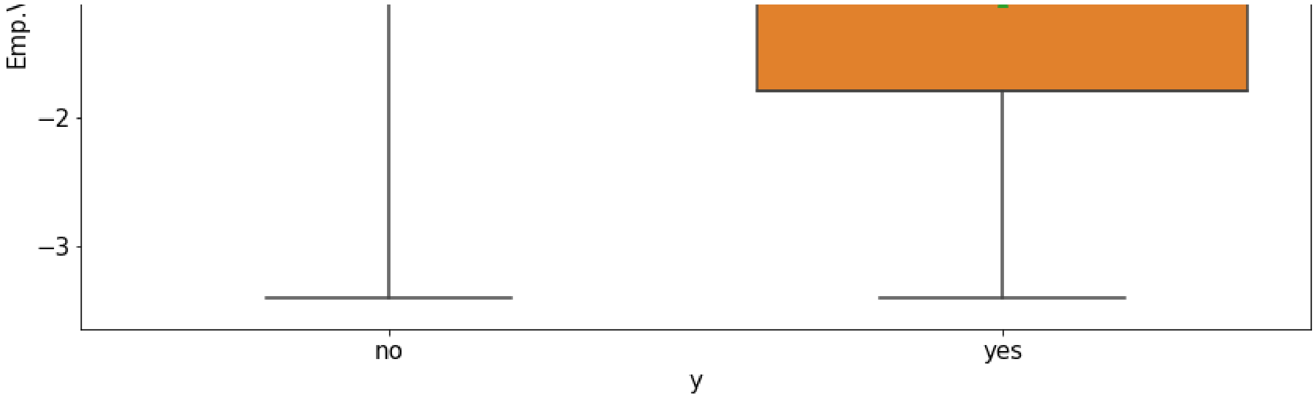


Variation in Emp.Var.Rate

In [30]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'emp.var.rate', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Emp.Var.Rate', fontsize=15)
bca1.set_title('Variation in Emp.Var.Rate', fontsize=15)
bca1.tick_params(labelsize=15)
```

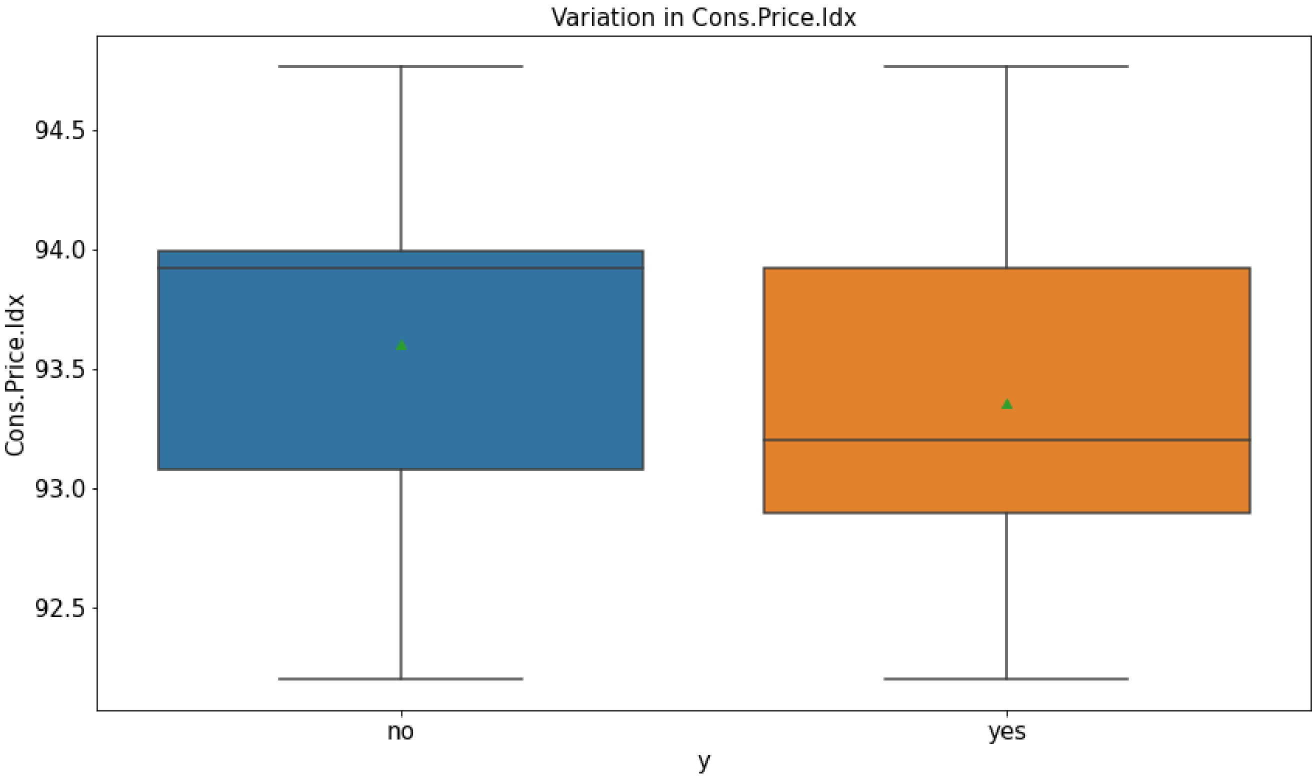




Variation in Cons.Price.Idx

In [31]:

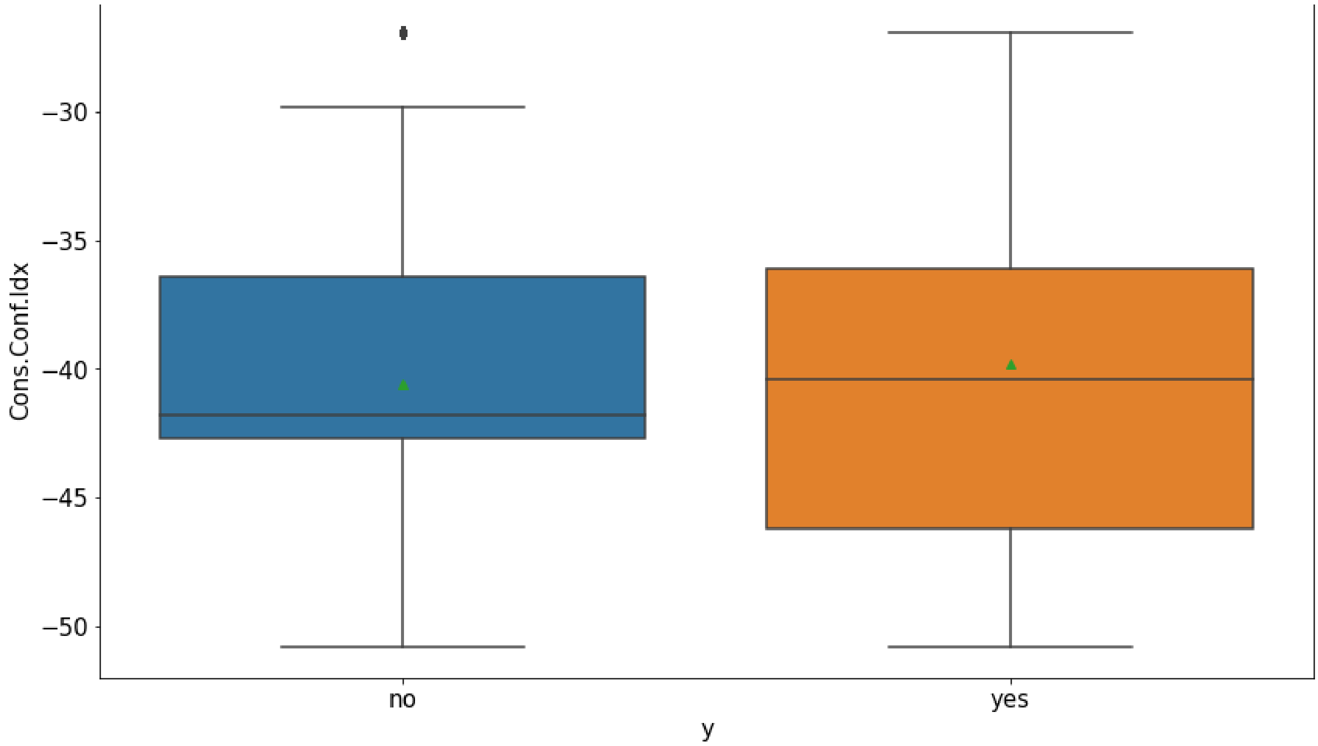
```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'cons.price.idx', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Cons.Price.Idx', fontsize=15)
bca1.set_title('Variation in Cons.Price.Idx', fontsize=15)
bca1.tick_params(labelsize=15)
```



Variation in Cons.Conf.Idx

In [32]:

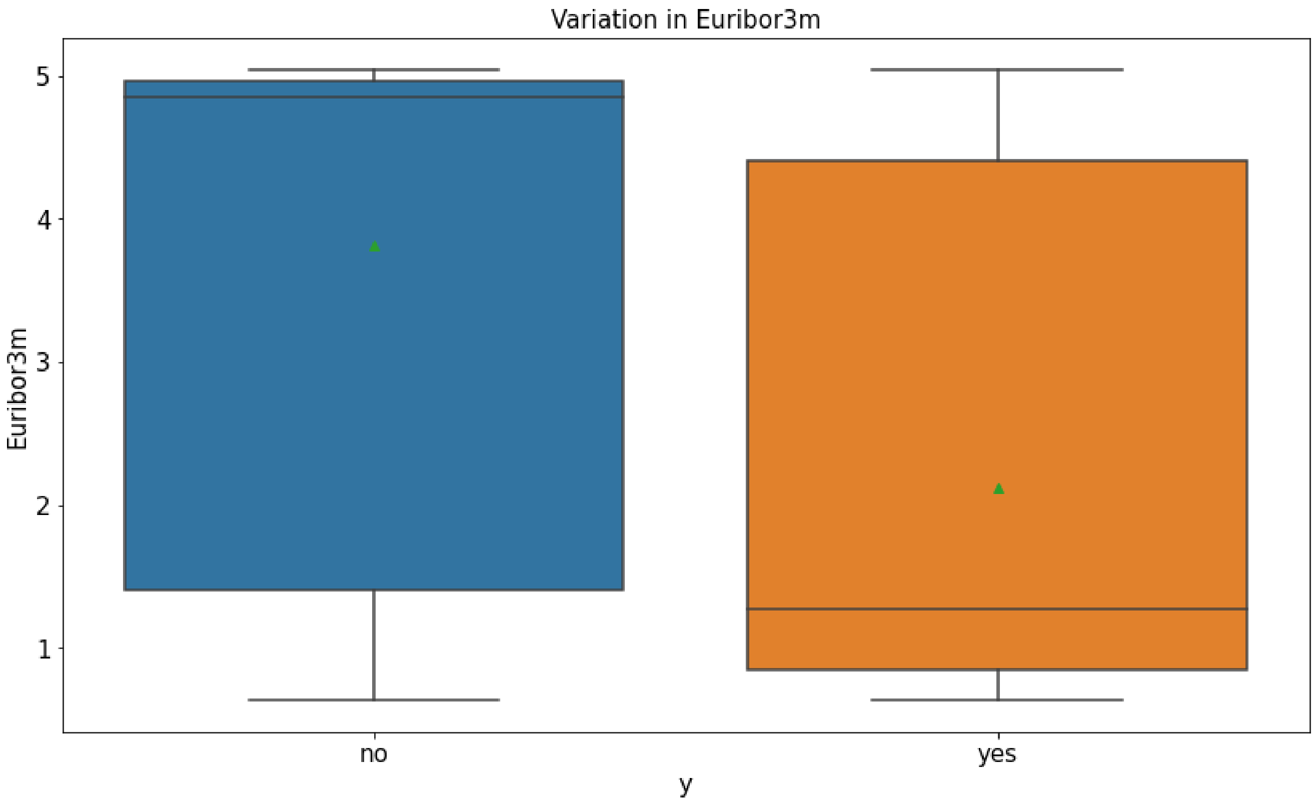
```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'cons.conf.idx', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Cons.Conf.Idx', fontsize=15)
bca1.set_title('Variation in Cons.Conf.Idx', fontsize=15)
bca1.tick_params(labelsize=15)
```



Variation in Euribor3m

In [33]:

```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'euribor3m', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Euribor3m', fontsize=15)
bca1.set_title('Variation in Euribor3m', fontsize=15)
bca1.tick_params(labelsize=15)
```

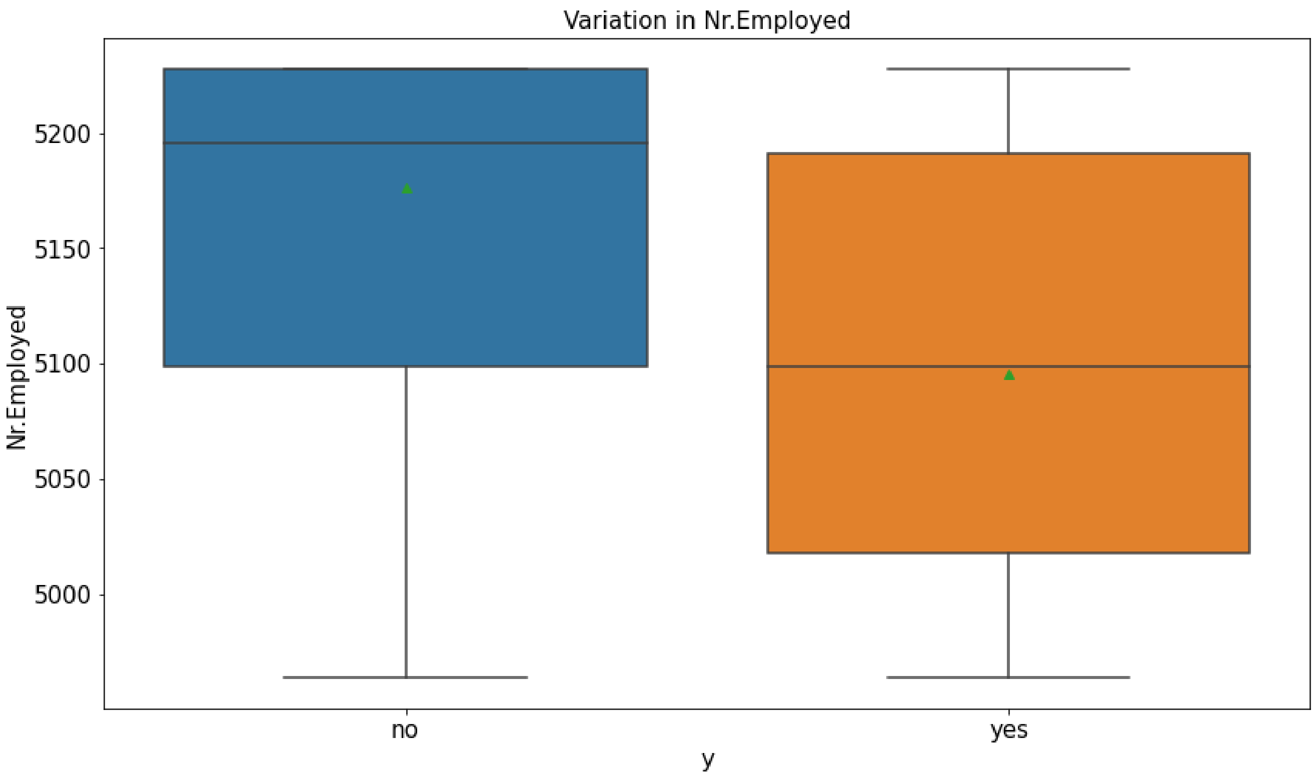


Variation in Nr Employed

Variation in Nr.Employed

In [34]:

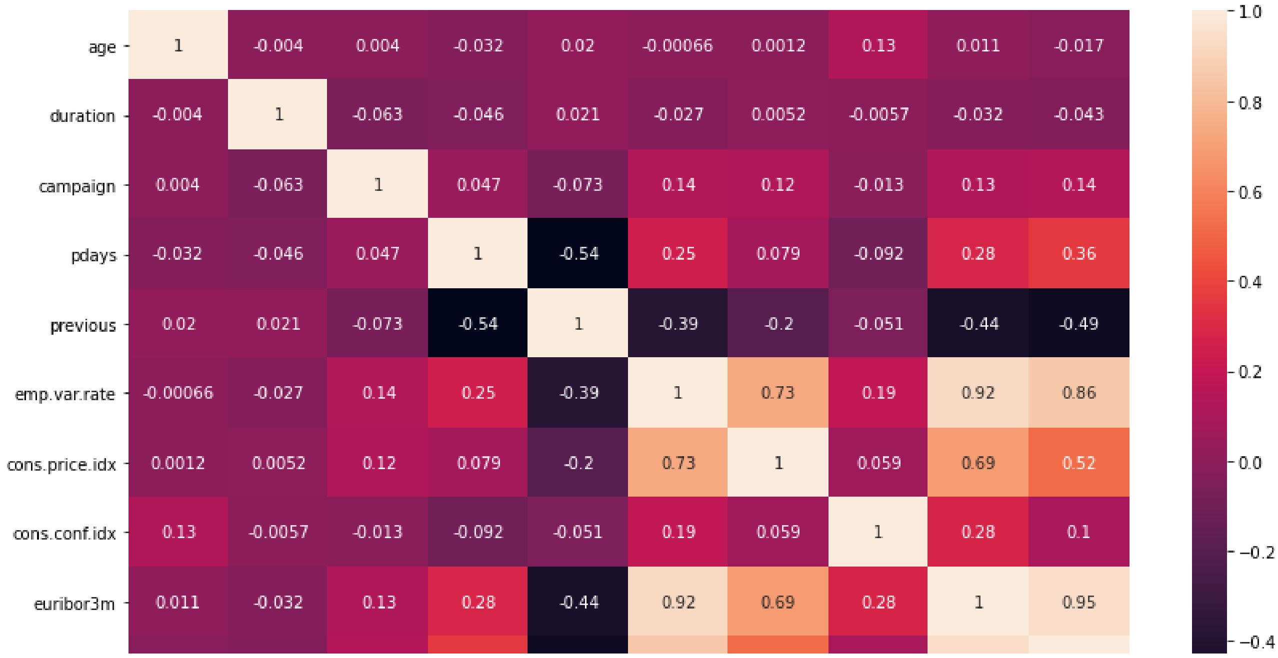
```
fig, bca = plt.subplots()
fig.set_size_inches(14, 8)
bca1 =sns.boxplot( x='y', y= 'nr.employed', data =bnk, showmeans=True)
bca1.set_xlabel('y', fontsize=15)
bca1.set_ylabel('Nr.Employed', fontsize=15)
bca1.set_title('Variation in Nr.Employed', fontsize=15)
bca1.tick_params(labelsize=15)
```

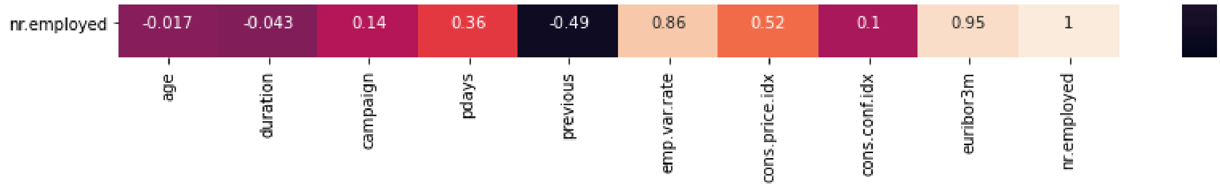


Heatmap depicting correlation between all numerical variables

In [35]:

```
plt.subplots(figsize=(14,8))
sns.heatmap(bnk.corr(), annot=True)
plt.show()
```





Encoding and storing target variable 'y'

We perform one-hot-encoding on target variable 'y' in bnk dataframe as it is categorical data. We store the result in a new variable 'y'.

In [36]:

```
y = pd.get_dummies(bnk['y'], columns = ['y'], prefix = ['y'], drop_first = True)
bnk.head()
```

Out[36]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	pou
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	2.570404	999.0	0.172596	non
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1.000000	999.0	0.000000	non
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1.000000	999.0	0.000000	non
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1.000000	999.0	0.000000	non
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1.000000	999.0	0.000000	non

5 rows × 21 columns

Creating a new dataframe 'bank_client'

We are creating the bank_client dataset to store information of bank clients. The attributes included are namely - age, job, marital, education, default, housing, loan

In [37]:

```
bank_client = bnk.iloc[: , 0:7]
bank_client.head()
```

Out[37]:

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

Exploring variables in bank_client

Age Count distribution

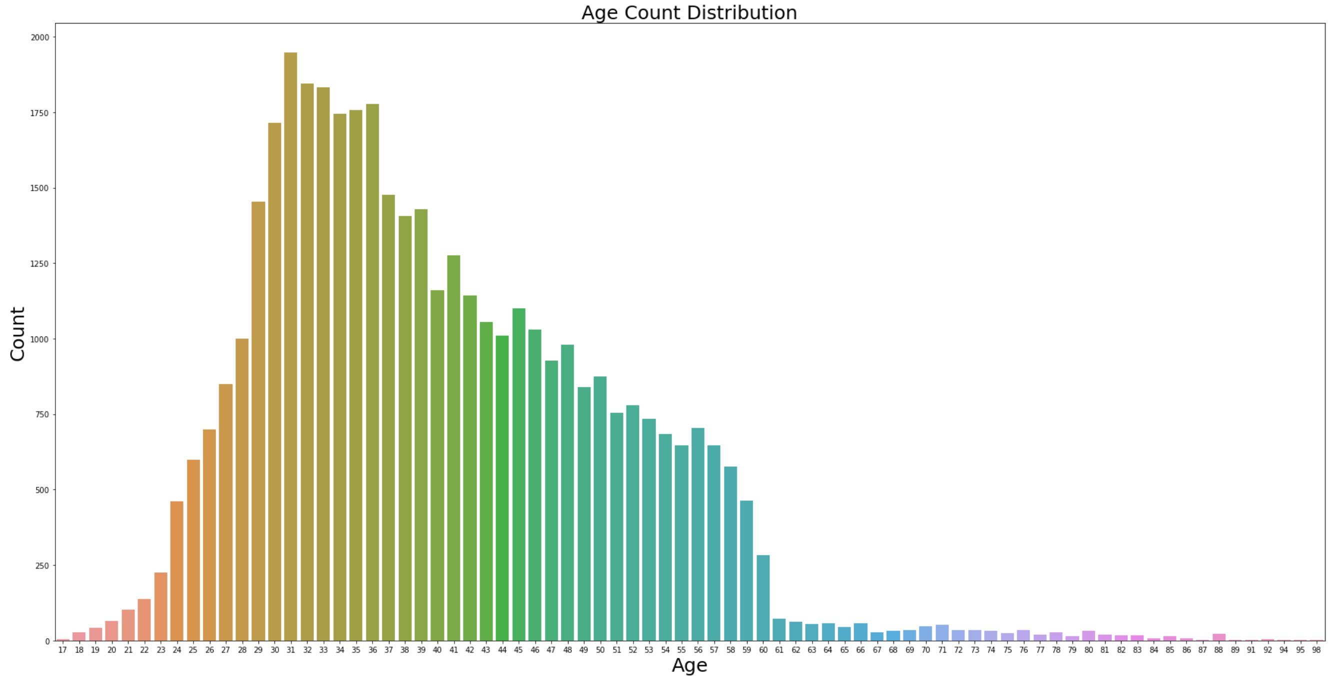
In [38]:

```
fig, bca = plt.subplots()
fig.set_size_inches(30, 15)
sns.countplot(x = 'age', data = bank_client)
bca.set_xlabel('Age', fontsize=25)
```

```
bca.set_ylabel('Count', fontsize=25)
bca.set_title('Age Count Distribution', fontsize=25)
```

Out[38]:

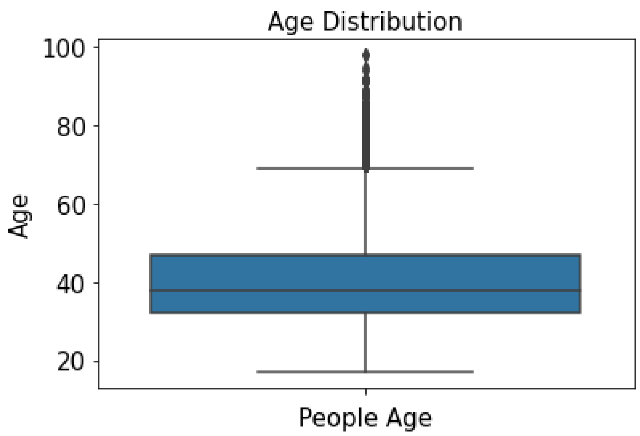
```
Text(0.5, 1.0, 'Age Count Distribution')
```



Age Distribution

In [39]:

```
bca1 =sns.boxplot( y=bank_client["age"] )
bca1.set_xlabel('People Age', fontsize=15)
bca1.set_ylabel('Age', fontsize=15)
bca1.set_title('Age Distribution', fontsize=15)
bca1.tick_params(labelsize=15)
```



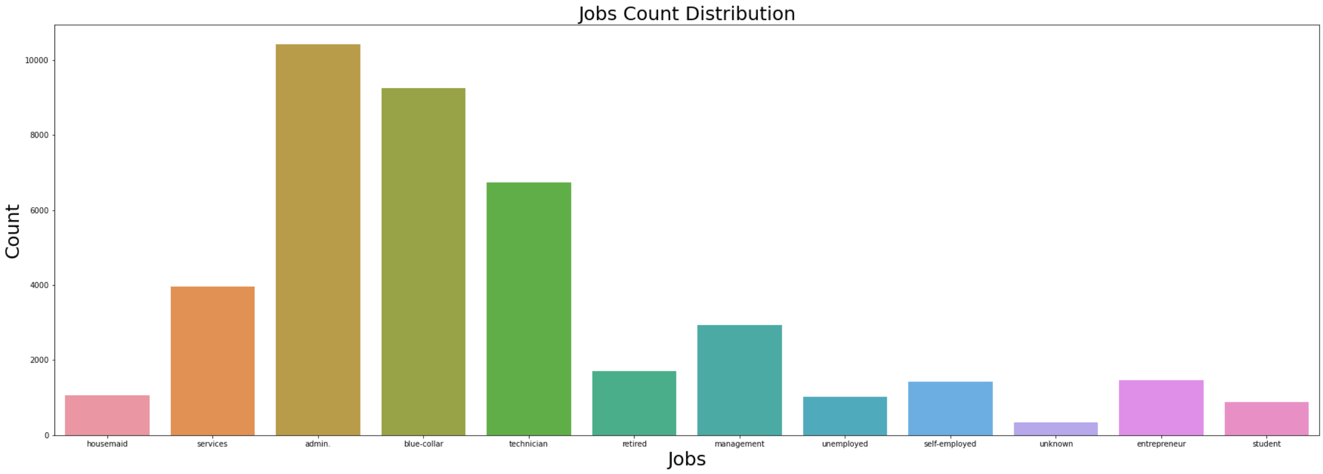
Jobs Count Distribution

In [40]:

```
fig, bca = plt.subplots()
fig.set_size_inches(30, 10)
sns.countplot(x = 'job', data = bank_client)
bca.set_xlabel('Jobs', fontsize=25)
bca.set_ylabel('Count', fontsize=25)
bca.set_title('Jobs Count Distribution', fontsize=25)
```

Out[40]:

```
Text(0.5, 1.0, 'Jobs Count Distribution')
```



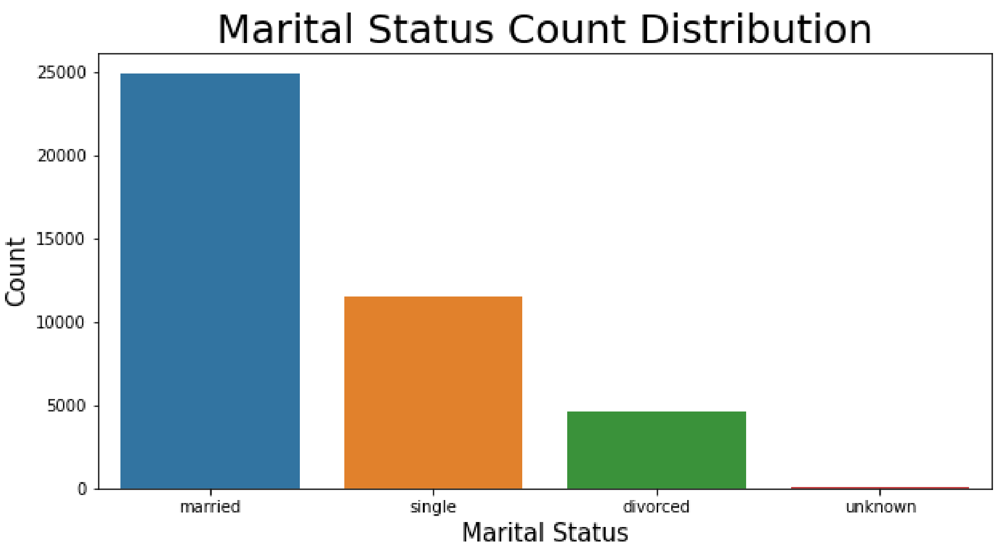
Marital Status Count Distribution

In [41]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'marital', data = bank_client)
bca.set_xlabel('Marital Status', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Marital Status Count Distribution', fontsize=25)
```

Out[41]:

```
Text(0.5, 1.0, 'Marital Status Count Distribution')
```



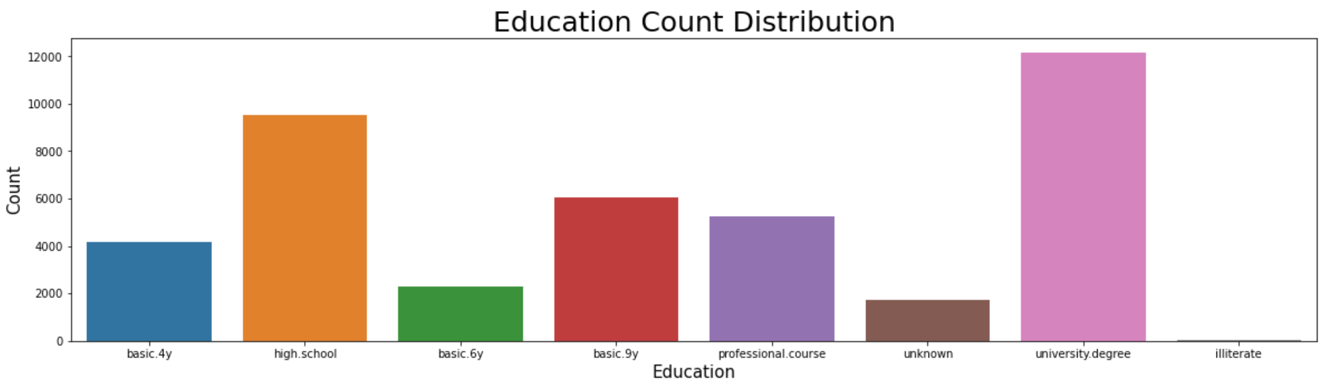
Education Count Distribution

In [42]:

```
fig, bca = plt.subplots()
fig.set_size_inches(20, 5)
sns.countplot(x = 'education', data = bank_client)
bca.set_xlabel('Education', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Education Count Distribution', fontsize=25)
```

Out[42]:

```
Text(0.5, 1.0, 'Education Count Distribution')
```



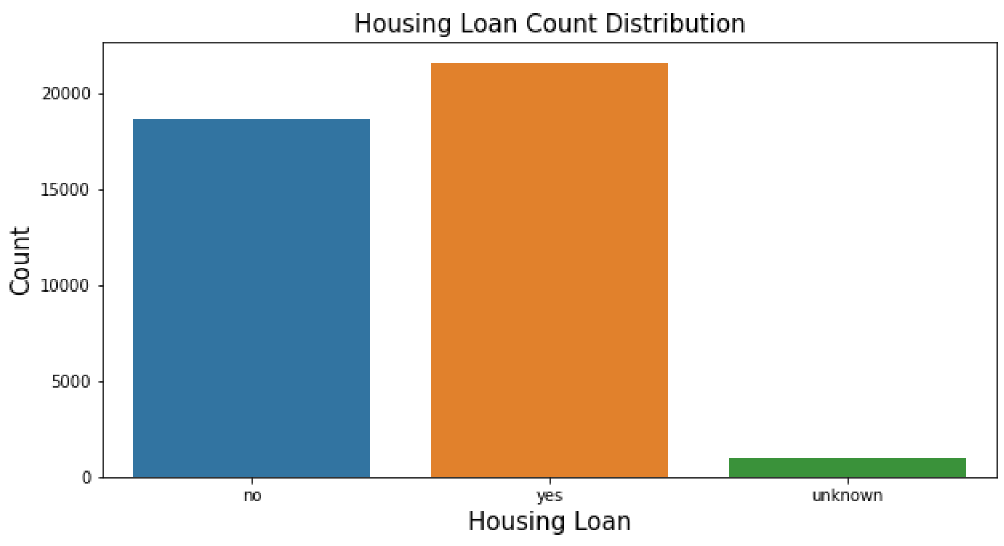
Housing Loan Count Distribution

In [43]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'housing', data = bank_client)
bca.set_xlabel('Housing Loan', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Housing Loan Count Distribution', fontsize=15)
```

Out[43]:

```
Text(0.5, 1.0, 'Housing Loan Count Distribution')
```



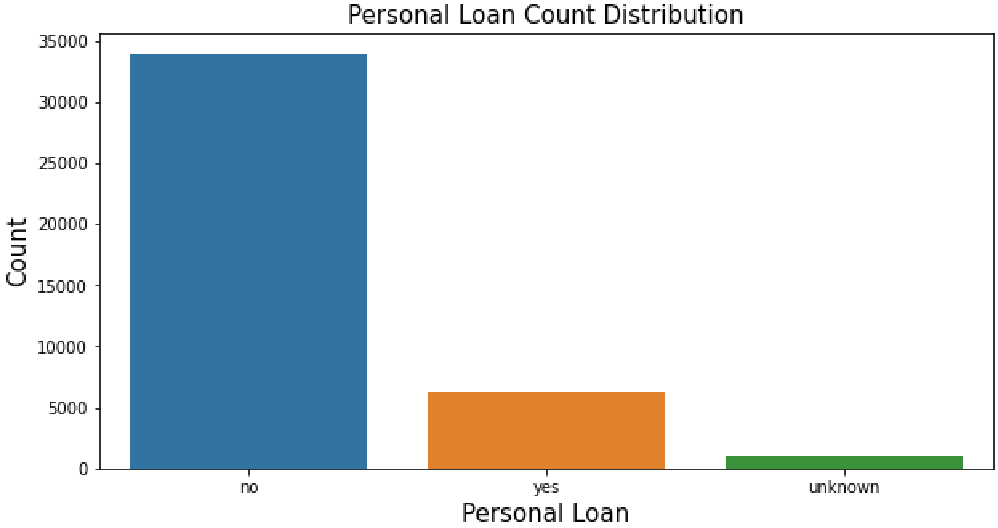
Personal Loan Count Distribution

In [44]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'loan', data = bank_client)
bca.set_xlabel('Personal Loan', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Personal Loan Count Distribution', fontsize=15)
```

Out[44]:

```
Text(0.5, 1.0, 'Personal Loan Count Distribution')
```



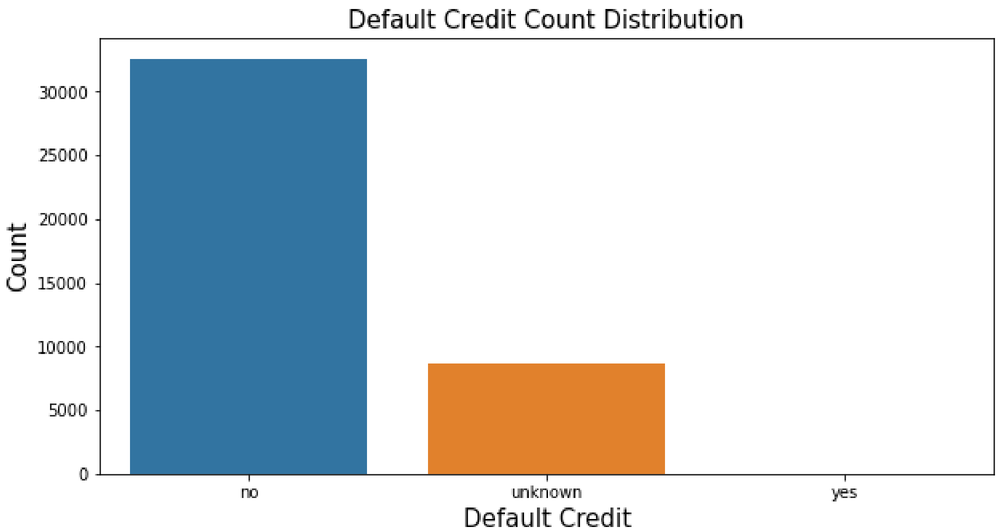
Default Credit Count Distribution

In [45]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'default', data = bank_client)
bca.set_xlabel('Default Credit', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Default Credit Count Distribution', fontsize=15)
```

Out[45]:

```
Text(0.5, 1.0, 'Default Credit Count Distribution')
```



Treating categorical variables

In [46]:

```
bank_client['job'].unique()
```

Out[46]:

```
array(['housemaid', 'services', 'admin.', 'blue-collar', 'technician',
       'retired', 'management', 'unemployed', 'self-employed', 'unknown',
       'entrepreneur', 'student'], dtype=object)
```

In [47]:

```
dummy = pd.get_dummies(bank_client['job'],prefix = 'Job_N')
print(dummy)
```

```
       Job_N_admin.  Job_N_blue-collar  Job_N_entrepreneur  Job_N_housemaid  \
0                 0                  0                   0                1   
1                 0                  0                   0                0   
2                 0                  0                   0                0   
3                 1                  0                   0                0   
4                 0                  0                   0                0   
...             ...                ...                 ...              ...   
41183             0                  0                   0                0   
41184             0                  1                   0                0   
41185             0                  0                   0                0   
41186             0                  0                   0                0   
41187             0                  0                   0                0   

       Job_N_management  Job_N_retired  Job_N_self-employed  Job_N_services  \
0                     0              0                    0               0   
1                     0              0                    0               1   
2                     0              0                    0               1   
3                     0              0                    0               0   
4                     0              0                    0               1   
...                 ...            ...                  ...             ...   
41183                 0              1                    0               0   
41184                 0              0                    0               0   
41185                 0              1                    0               0   
41186                 0              0                    0               0   
41187                 0              1                    0               0   

       Job_N_student  Job_N_technician  Job_N_unemployed  Job_N_unknown  
0                  0                 0                 0              0  
1                  0                 0                 0              0  
2                  0                 0                 0              0  
3                  0                 0                 0              0  
4                  0                 0                 0              0  
...              ...               ...               ...            ...  
41183              0                 0                 0              0  
41184              0                 0                 0              0  
41185              0                 0                 0              0  
41186              0                 1                 0              0  
41187              0                 0                 0              0  

[41164 rows x 12 columns]
```

In [48]:

```
bank_client = bank_client.join(dummy)
bank_client
```

Out[48]:

	age	job	marital	education	default	housing	loan	Job_N_admin.	Job_N_blue-collar	Job_N_entrepreneur	Job_N_I
0	56	housemaid	married	basic.4y	no	no	no	0	0	0	
1	57	services	married	high.school	unknown	no	no	0	0	0	
2	37	services	married	high.school	no	yes	no	0	0	0	
3	40	admin.	married	basic.6y	no	no	no	1	0	0	
4	56	services	married	high.school	no	no	yes	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	0	0	0	
41184	46	blue-collar	married	professional.course	no	no	no	0	1	0	
41185	56	retired	married	university.degree	no	yes	no	0	0	0	
41186	44	technician	married	professional.course	no	no	no	0	0	0	
41187	74	retired	married	professional.course	no	yes	no	0	0	0	

41164 rows × 19 columns

In [49]:

```
bank_client['marital'].unique()
```

Out[49]:

```
array(['married', 'single', 'divorced', 'unknown'], dtype=object)
```

In [50]:

```
lc=LabelEncoder()
bank_client['Marital_N']=lc.fit_transform(bank_client['marital'])
bank_client
```

Out[50]:

	age	job	marital	education	default	housing	loan	Job_N_admin.	Job_N_blue-collar	Job_N_entrepreneur	Job_N_I
0	56	housemaid	married	basic.4y	no	no	no	0	0	0	
1	57	services	married	high.school	unknown	no	no	0	0	0	
2	37	services	married	high.school	no	yes	no	0	0	0	
3	40	admin.	married	basic.6y	no	no	no	1	0	0	
4	56	services	married	high.school	no	no	yes	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	0	0	0	
41184	46	blue-collar	married	professional.course	no	no	no	0	1	0	
41185	56	retired	married	university.degree	no	yes	no	0	0	0	
41186	44	technician	married	professional.course	no	no	no	0	0	0	
41187	74	retired	married	professional.course	no	yes	no	0	0	0	

41164 rows × 20 columns

In [51]:

```
bank_client['education'].unique()
```

Out[51]:

```
array(['basic.4y', 'high.school', 'basic.6y', 'basic.9y',
       'professional.course', 'unknown', 'university.degree',
       'illiterate'], dtype=object)
```

In [52]:

```
bank_client=pd.concat((bank_client,pd.get_dummies(bank_client['education'])),axis=1)
```

In [53]:

```
bank_client['default'].unique()
```

Out[53]:

```
array(['no', 'unknown', 'yes'], dtype=object)
```

In [54]:

```
bank_client['housing'].unique()
```

Out[54]:

```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [55]:

```
bank_client['loan'].unique()
```

Out[55]:

```
array(['no', 'yes', 'unknown'], dtype=object)
```

In [56]:

```
lc=LabelEncoder()
bank_client['Default_N']=lc.fit_transform(bank_client['default'])
```

In [57]:

```
lc=LabelEncoder()
bank_client['Housing_N']=lc.fit_transform(bank_client['housing'])
```

In [58]:

```
lc=LabelEncoder()
bank_client['Loan_N']=lc.fit_transform(bank_client['loan'])
```

In [59]:

```
bank_client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  41164 non-null  int64
 1   job                  41164 non-null  object
 2   marital              41164 non-null  object
 3   education            41164 non-null  object
 4   default              41164 non-null  object
 5   housing              41164 non-null  object
 6   loan                 41164 non-null  object
 7   Job_N_admin.         41164 non-null  uint8
 8   Job_N_blue-collar    41164 non-null  uint8
 9   Job_N_entrepreneur   41164 non-null  uint8
 10  Job_N_housemaid      41164 non-null  uint8
 11  Job_N_management     41164 non-null  uint8
 12  Job_N_retired        41164 non-null  uint8
 13  Job_N_self-employed  41164 non-null  uint8
 14  Job_N_services       41164 non-null  uint8
 15  Job_N_student        41164 non-null  uint8
 16  Job_N_technician     41164 non-null  uint8
 17  Job_N_unemployed     41164 non-null  uint8
 18  Job_N_unknown        41164 non-null  uint8
 19  Marital_N            41164 non-null  int32
 20  basic.4y             41164 non-null  uint8
 21  basic.6y             41164 non-null  uint8
 22  basic.9y             41164 non-null  uint8
 23  high.school          41164 non-null  uint8
 24  illiterate           41164 non-null  uint8
 25  professional.course  41164 non-null  uint8
 26  university.degree    41164 non-null  uint8
 27  unknown              41164 non-null  uint8
 28  Default_N            41164 non-null  int32
 29  Housing_N            41164 non-null  int32
 30  Loan_N               41164 non-null  int32
dtypes: int32(4), int64(1), object(6), uint8(20)
memory usage: 5.2+ MB
```

In [60]:

```
bank_client = bank_client.drop(['job', 'marital', 'education', 'housing', 'default', 'loan'], axis
= 1)
```

In [61]:

```
bank_client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  41164 non-null  int64
 1   Job_N_admin.         41164 non-null  uint8
 2   Job_N_blue-collar    41164 non-null  uint8
 3   Job_N_entrepreneur   41164 non-null  uint8
 4   Job_N_housemaid      41164 non-null  uint8
 5   Job_N_management     41164 non-null  uint8
 6   Job_N_retired        41164 non-null  uint8
 7   Job_N_self-employed  41164 non-null  uint8
 8   Job_N_services       41164 non-null  uint8
 9   Job_N_student        41164 non-null  uint8
 10  Job_N_technician     41164 non-null  uint8
 11  Job_N_unemployed     41164 non-null  uint8
 12  Job_N_unknown        41164 non-null  uint8
 13  Marital_N            41164 non-null  int32
 14  basic.4y             41164 non-null  uint8
 15  basic.6y             41164 non-null  uint8
 16  basic.9y             41164 non-null  uint8
 17  high.school          41164 non-null  uint8
 18  illiterate           41164 non-null  uint8
 19  professional.course  41164 non-null  uint8
 20  university.degree    41164 non-null  uint8
 21  unknown              41164 non-null  uint8
 22  Default_N            41164 non-null  int32
 23  Housing_N            41164 non-null  int32
 24  Loan_N               41164 non-null  int32
dtypes: int32(4), int64(1), uint8(20)
memory usage: 3.3 MB
```

In [62]:

```
bank_client['age'] = bank_client['age'].astype(int)
```

In [63]:

```
bank_client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 25 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  41164 non-null  int32
 1   Job_N_admin.         41164 non-null  uint8
 2   Job_N_blue-collar    41164 non-null  uint8
 3   Job_N_entrepreneur   41164 non-null  uint8
 4   Job_N_housemaid      41164 non-null  uint8
 5   Job_N_management     41164 non-null  uint8
 6   Job_N_retired        41164 non-null  uint8
 7   Job_N_self-employed  41164 non-null  uint8
 8   Job_N_services       41164 non-null  uint8
 9   Job_N_student        41164 non-null  uint8
 10  Job_N_technician     41164 non-null  uint8
 11  Job_N_unemployed     41164 non-null  uint8
 12  Job_N_unknown        41164 non-null  uint8
 13  Marital_N            41164 non-null  int32
 14  basic.4y             41164 non-null  uint8
 15  basic.6y             41164 non-null  uint8
 16  basic.9y             41164 non-null  uint8
 17  high.school          41164 non-null  uint8
 18  illiterate           41164 non-null  uint8
 19  professional.course  41164 non-null  uint8
 20  university.degree    41164 non-null  uint8
 21  unknown              41164 non-null  uint8
 22  Default_N            41164 non-null  int32
```

 23 Housing_N   41164 non-null int32
 24 Loan_N   41164 non-null int32
dtypes: int32(5), uint8(20)
memory usage: 3.1 MB

In [64]:

```python
def age(dataframe):
    dataframe.loc[dataframe['age'] <= 32, 'age'] = 1
    dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] <= 47), 'age'] = 2
    dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] <= 70), 'age'] = 3
    dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] <= 98), 'age'] = 4

    return dataframe

age(bank_client) ;
```

In [65]:

```python
bank_client.head()
```

Out[65]:

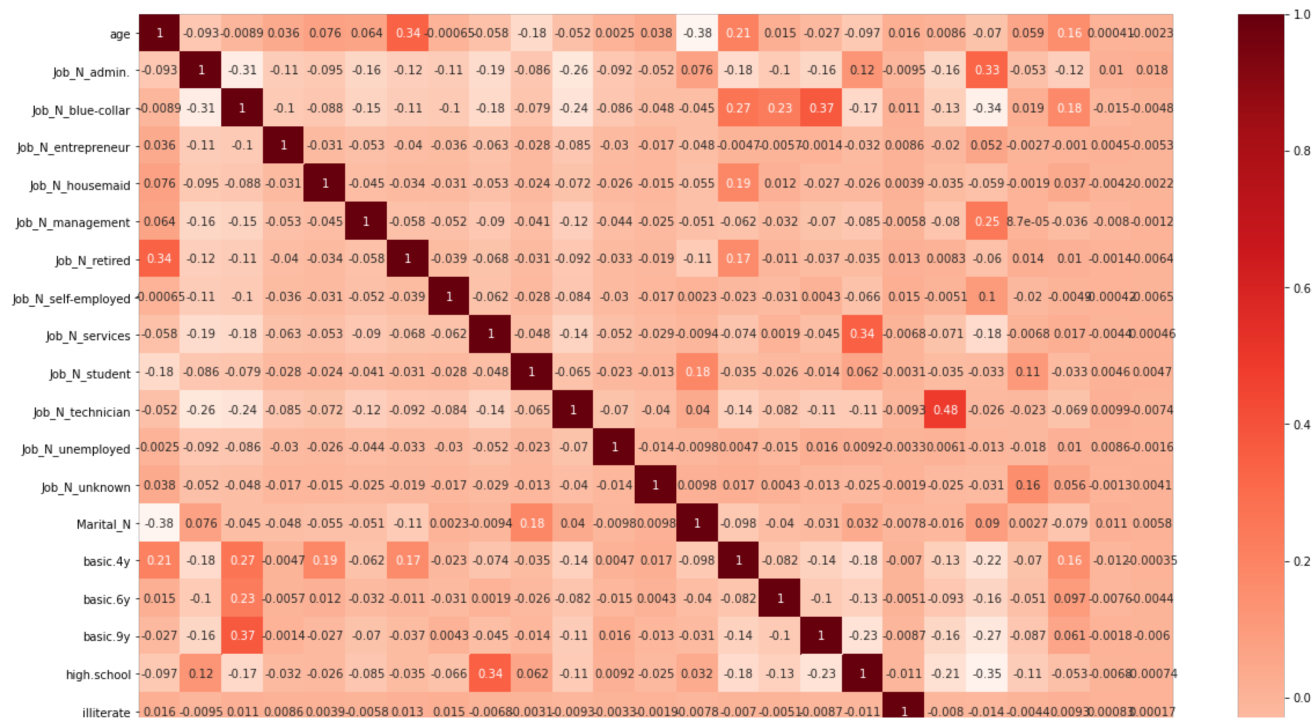
	age	Job_N_admin.	Job_N_blue-collar	Job_N_entrepreneur	Job_N_housemaid	Job_N_management	Job_N_retired	Job_N_self-employed	Job_N_s
0	3	0	0	0	1	0	0	0	
1	3	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	
3	2	1	0	0	0	0	0	0	
4	3	0	0	0	0	0	0	0	

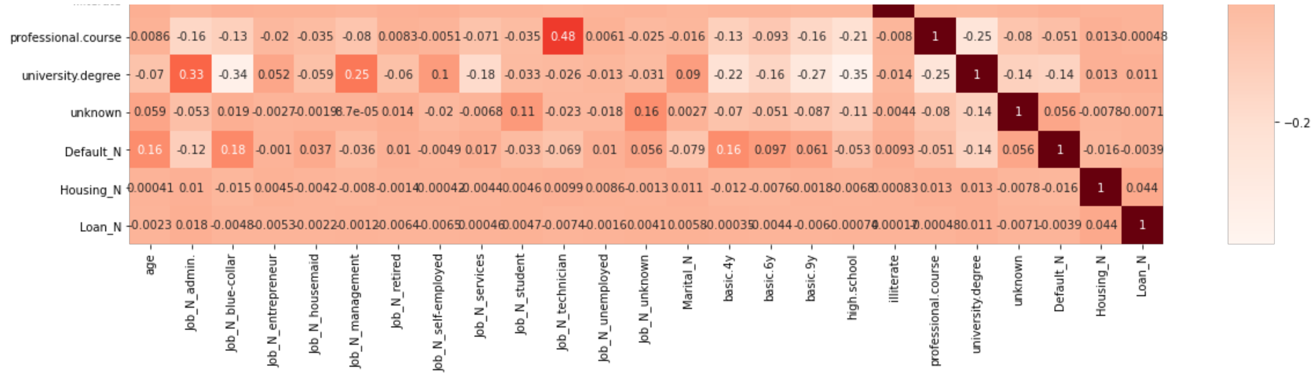
5 rows × 25 columns

Correlation between variables in bank_client dataset

In [66]:

```python
plt.figure(figsize=(20,15))
cor = bank_client.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```





Creating new dataset 'other_attr'

We are now creating a dataset to store the attributes - contact, month, day_of_week and duration.

In [67]:

```
other_attr = bnk.iloc[: , 7:11]
other_attr.head()
```

Out[67]:

	contact	month	day_of_week	duration
0	telephone	may	mon	261.0
1	telephone	may	mon	149.0
2	telephone	may	mon	226.0
3	telephone	may	mon	151.0
4	telephone	may	mon	307.0

Check for null values

In [68]:

```
other_attr.isnull().sum()
```

Out[68]:

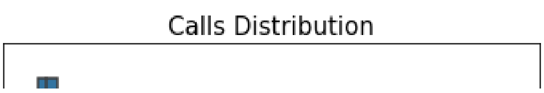
```
contact        0
month          0
day_of_week    0
duration       0
dtype: int64
```

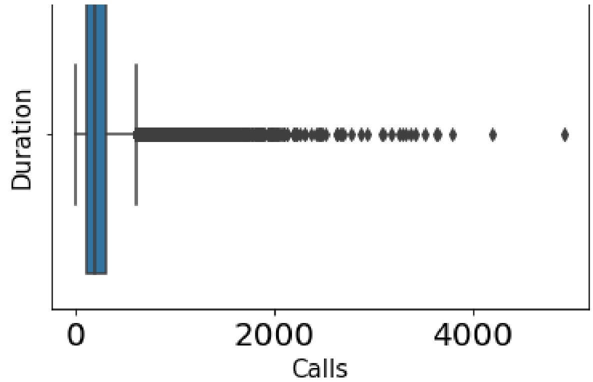
Exploring the attributes in 'other_attr'

Calls duration

In [69]:

```
dur = sns.boxplot(x = 'duration', data = other_attr)
dur.set_xlabel('Calls', fontsize=15)
dur.set_ylabel('Duration', fontsize=15)
dur.set_title('Calls Distribution', fontsize=15)
dur.tick_params(labelsize=20)
```





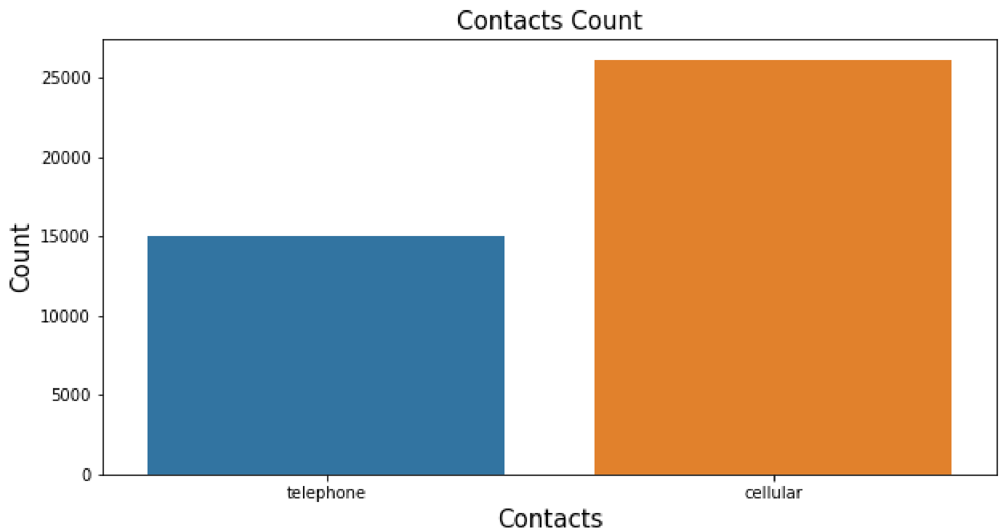
Contacts Count

In [70]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'contact', data = other_attr)
bca.set_xlabel('Contacts', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Contacts Count', fontsize=15)
```

Out[70]:

```
Text(0.5, 1.0, 'Contacts Count')
```



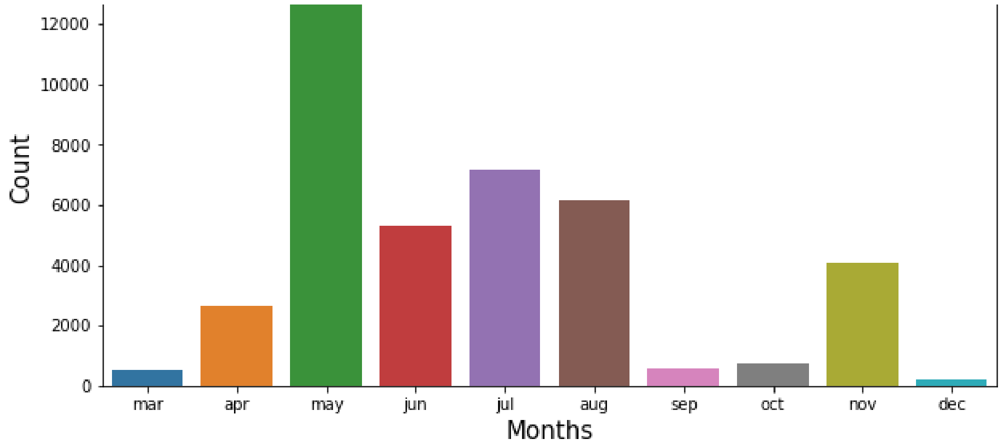
Months Count

In [71]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'month', data = other_attr, order = ['mar', 'apr', 'may', 'jun', 'jul', 'aug', 's
ep', 'oct', 'nov', 'dec'])
bca.set_xlabel('Months', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('Months Count', fontsize=15)
```

Out[71]:

```
Text(0.5, 1.0, 'Months Count')
```



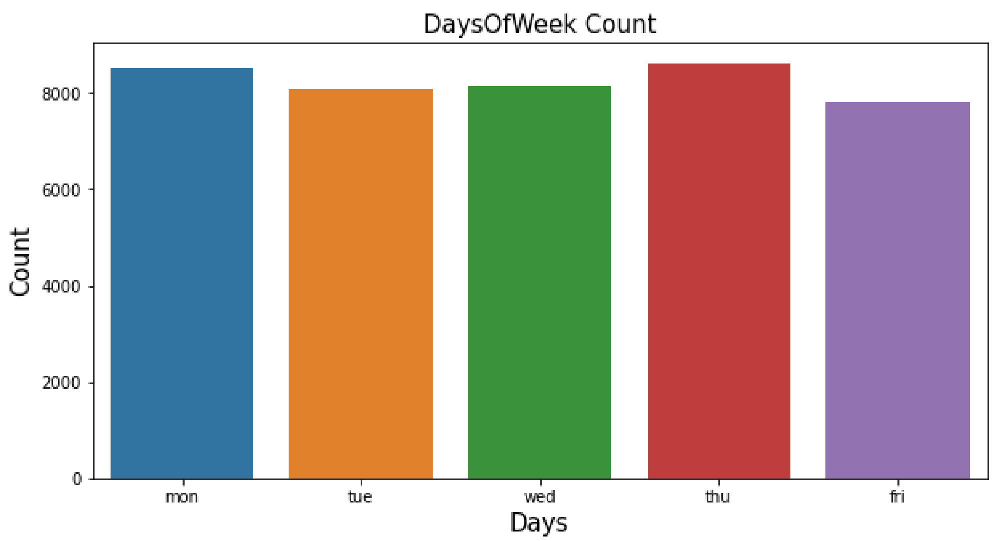
Days Of Week Count

In [72]:

```
fig, bca = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'day_of_week', data = other_attr)
bca.set_xlabel('Days', fontsize=15)
bca.set_ylabel('Count', fontsize=15)
bca.set_title('DaysOfWeek Count', fontsize=15)
```

Out[72]:

```
Text(0.5, 1.0, 'DaysOfWeek Count')
```



Treating categorical variables

In [73]:

```
other_attr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   contact      41164 non-null  object
 1   month        41164 non-null  object
 2   day_of_week  41164 non-null  object
 3   duration     41164 non-null  float64
dtypes: float64(1), object(3)
memory usage: 2.8+ MB
```

In [74]:

```
other_attr['contact'].unique()
```

Out[74]:

```
array(['telephone', 'cellular'], dtype=object)
```

In [75]:

```
other_attr['contact'] = other_attr['contact'].map({'telephone':1, 'cellular':2}).astype(int)
```

In [76]:

```
other_attr['month'].unique()
```

Out[76]:

```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'mar', 'apr',
       'sep'], dtype=object)
```

In [77]:

```
other_attr["month"] = other_attr["month"].str.capitalize()
```

In [78]:

```
other_attr["month"] = pd.to_datetime(other_attr.month, format='%b', errors='coerce').dt.month
other_attr = other_attr.sort_values(by="month")
```

In [79]:

```
other_attr['month'].unique()
```

Out[79]:

```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

In [80]:

```
other_attr['day_of_week'].unique()
```

Out[80]:

```
array(['tue', 'mon', 'thu', 'wed', 'fri'], dtype=object)
```

In [81]:

```
lc=LabelEncoder()
other_attr['day_of_week']=lc.fit_transform(other_attr['day_of_week'])
```

In [82]:

```
other_attr['day_of_week'].unique()
```

Out[82]:

```
array([3, 1, 2, 4, 0])
```

In [83]:

```
other_attr['duration'] = other_attr['duration'].astype(int)
```

Creating new dataset 'cont_attr'

Here we are creating a new dataset for the social and economic context attributes, which are - emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed.

In [84]:

```
cont_attr= bnk.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employe
d']]
cont_attr.head()
```

Out[84]:

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	1.100000	93.994	-36.4	4.857	5191.0
1	0.079252	93.994	-36.4	4.857	5191.0
2	1.100000	93.994	-36.4	4.857	5191.0
3	1.100000	93.994	-36.4	4.857	5191.0
4	1.100000	93.994	-36.4	4.857	5191.0

Creating a dataset 'remain_attr'

This dataset contains all the remaining attributes (y excluded). These are - campaign, pdays, previous, poutcome

In [85]:

```
remain_attr = bnk.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
remain_attr.head()
```

Out[85]:

	campaign	pdays	previous	poutcome
0	2.570404	999.0	0.172596	nonexistent
1	1.000000	999.0	0.000000	nonexistent
2	1.000000	999.0	0.000000	nonexistent
3	1.000000	999.0	0.000000	nonexistent
4	1.000000	999.0	0.000000	nonexistent

Treating the categorical attributes

In [86]:

```
remain_attr['poutcome'].unique()
```

Out[86]:

```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

In [87]:

```
remain_attr['poutcome'] = remain_attr['poutcome'].map({'nonexistent':1, 'failure':2, 'success':3}).
astype(int)
```

Creating the 'final_bank' dataset

We will now merge/concat all the above datasets that we created and curated as per need into one final dataset for our analysis

analysis.

In [88]:

```
final_bank= pd.concat([bank_client, other_attr, cont_attr, remain_attr], axis = 1)
```

Exploring the dataset

In [89]:

```
final_bank.shape
```

Out[89]:

```
(41164, 38)
```

In [90]:

```
final_bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41164 entries, 0 to 41187
Data columns (total 38 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   age                 41164 non-null  int32
 1   Job_N_admin.        41164 non-null  uint8
 2   Job_N_blue-collar   41164 non-null  uint8
 3   Job_N_entrepreneur  41164 non-null  uint8
 4   Job_N_housemaid     41164 non-null  uint8
 5   Job_N_management    41164 non-null  uint8
 6   Job_N_retired       41164 non-null  uint8
 7   Job_N_self-employed 41164 non-null  uint8
 8   Job_N_services      41164 non-null  uint8
 9   Job_N_student       41164 non-null  uint8
 10  Job_N_technician    41164 non-null  uint8
 11  Job_N_unemployed    41164 non-null  uint8
 12  Job_N_unknown       41164 non-null  uint8
 13  Marital_N           41164 non-null  int32
 14  basic.4y            41164 non-null  uint8
 15  basic.6y            41164 non-null  uint8
 16  basic.9y            41164 non-null  uint8
 17  high.school         41164 non-null  uint8
 18  illiterate          41164 non-null  uint8
 19  professional.course 41164 non-null  uint8
 20  university.degree   41164 non-null  uint8
 21  unknown             41164 non-null  uint8
 22  Default_N           41164 non-null  int32
 23  Housing_N           41164 non-null  int32
 24  Loan_N              41164 non-null  int32
 25  contact             41164 non-null  int32
 26  month               41164 non-null  int64
 27  day_of_week         41164 non-null  int32
 28  duration            41164 non-null  int32
 29  emp.var.rate        41164 non-null  float64
 30  cons.price.idx      41164 non-null  float64
 31  cons.conf.idx       41164 non-null  float64
 32  euribor3m           41164 non-null  float64
 33  nr.employed         41164 non-null  float64
 34  campaign            41164 non-null  float64
 35  pdays               41164 non-null  float64
 36  previous            41164 non-null  float64
 37  poutcome            41164 non-null  int32
dtypes: float64(8), int32(9), int64(1), uint8(20)
memory usage: 5.3 MB
```

In [91]:

```
final_bank.isna().sum()
```

Out[91]:

```
age                     0
Job_N_admin.            0
Job_N_blue-collar       0
Job_N_entrepreneur      0
Job_N_housemaid         0
Job_N_management        0
Job_N_retired           0
Job_N_self-employed     0
Job_N_services          0
Job_N_student           0
Job_N_technician        0
Job_N_unemployed        0
Job_N_unknown           0
Marital_N               0
basic.4y                0
basic.6y                0
basic.9y                0
high.school             0
illiterate              0
professional.course     0
university.degree       0
unknown                 0
Default_N               0
Housing_N               0
Loan_N                  0
contact                 0
month                   0
day_of_week             0
duration                0
emp.var.rate            0
cons.price.idx          0
cons.conf.idx           0
euribor3m               0
nr.employed             0
campaign                0
pdays                   0
previous                0
poutcome                0
dtype: int64
```

In [92]:

```
final_bank['campaign'].unique()
```

Out[92]:

```
array([ 2.57040373,  1.        ,  2.        ,  3.        ,  4.        ,
        5.        ,  6.        ,  7.        ,  8.        ,  9.        ,
       10.        , 11.        , 12.        , 13.        , 19.        ,
       18.        , 23.        , 14.        , 22.        , 25.        ,
       16.        , 17.        , 15.        , 20.        , 56.        ,
       42.        , 28.        , 26.        , 27.        , 32.        ,
       21.        , 24.        , 29.        , 31.        , 30.        ,
       35.        , 41.        , 37.        , 40.        , 33.        ,
       34.        , 43.        ])
```

In [93]:

```
final_bank['campaign'].fillna(final_bank['campaign'].mean(),inplace=True)
```

In [94]:

```
final_bank.isna().sum()
```

Out[94]:

```
age                     0
Job_N_admin.            0
Job_N_blue-collar       0
Job_N_entrepreneur      0
Job_N_housemaid         0
Job_N_management        0
Job_N_retired           0
```

```
Job_N_self-employed    0
Job_N_services         0
Job_N_student          0
Job_N_technician       0
Job_N_unemployed       0
Job_N_unknown          0
Marital_N              0
basic.4y               0
basic.6y               0
basic.9y               0
high.school            0
illiterate             0
professional.course    0
university.degree      0
unknown                0
Default_N              0
Housing_N              0
Loan_N                 0
contact                0
month                  0
day_of_week            0
duration               0
emp.var.rate           0
cons.price.idx         0
cons.conf.idx          0
euribor3m              0
nr.employed            0
campaign               0
pdays                  0
previous               0
poutcome               0
dtype: int64
```

In [95]:

```
final_bank.describe()
```

Out[95]:

	age	Job_N_admin.	Job_N_blue-collar	Job_N_entrepreneur	Job_N_housemaid	Job_N_management	Job_N_retired	Job_I emp
count	41164.000000	41164.000000	41164.000000	41164.000000	41164.000000	41164.000000	41164.000000	41164.(
mean	1.978598	0.253037	0.224759	0.035371	0.025751	0.071033	0.041687	0.(
std	0.735708	0.434757	0.417429	0.184717	0.158392	0.256883	0.199875	0.
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
25%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
50%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
75%	2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
max	4.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.(

8 rows × 38 columns

Splitting the data

We already have our target variable stored in 'y' from the beginning. Also, we have seperately curated our final_bank dataset. So, it does not contain our target variable y from our original dataset.

In [96]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(final_bank,y, test_size = 0.2, random_state = 0
)
```

In [97]:

```
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

In [98]:

```
X_train.head()
```

Out[98]:

	age	Job_N_admin.	Job_N_blue-collar	Job_N_entrepreneur	Job_N_housemaid	Job_N_management	Job_N_retired	Job_N_self-employed	Job
20018	2	0	0	0	0	0	0	0	
39695	1	1	0	0	0	0	0	0	
17238	3	0	0	0	1	0	0	0	
5924	3	0	1	0	0	0	0	0	
34656	2	1	0	0	0	0	0	0	

5 rows × 38 columns

Scaling the data

In our final_bank data, we can see that the minimum and maimum value ranges from quite high to quite low values. For this reason, we are scaling our data with StandardScaler. We do so to scale our features centred around the zero and have unit variance.

In [99]:

```
#train-test split
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Voting Classifier

In [100]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier
```

Hard voting

In [101]:

```
log_reg_clf = LogisticRegression(random_state= 0, C = 100, max_iter = 1000)
log_reg_clf.fit(X_train, y_train)

dtree_clf = DecisionTreeClassifier(max_depth = 1, random_state = 0)
dtree_clf.fit(X_train, y_train)

svc_clf = SVC(C = 0.1, gamma = 0.01, probability = True, random_state= 0)
svc_clf.fit(X_train, y_train)

hard_voting_clf = VotingClassifier(estimators=[('lr', log_reg_clf), ('dt', dtree_clf), ('svc',
svc_clf)], voting='hard')
hard_voting_clf.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
for clf in (log_reg_clf, dtree_clf, svc_clf, hard_voting_clf):
    clf.fit(X_train, y_train)
```

```
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, '%.4f'%accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.9093
DecisionTreeClassifier 0.8876
SVC 0.8987
VotingClassifier 0.9014
```

Soft Voting

In [102]:

```
log_reg_clf = LogisticRegression(random_state= 0, C = 100, max_iter = 1000)
log_reg_clf.fit(X_train, y_train)

dtree_clf = DecisionTreeClassifier(max_depth = 1, random_state = 0)
dtree_clf.fit(X_train, y_train)

svc_clf = SVC(C = 0.1, gamma = 0.01, probability = True, random_state= 0)
svc_clf.fit(X_train, y_train)

soft_voting_clf = VotingClassifier(estimators=[('lr', log_reg_clf), ('dt', dtree_clf), ('svc',
svc_clf)], voting='soft')
soft_voting_clf.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
for clf in (log_reg_clf, dtree_clf, svc_clf, soft_voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, '%.4f'%accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.9093
DecisionTreeClassifier 0.8876
SVC 0.8987
VotingClassifier 0.9030
```

Bagging

Bagging for Decision Tree Classifier

In [104]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

dtree_clf = DecisionTreeClassifier(random_state=0)
dtree_bag_clf = BaggingClassifier(dtree_clf, n_estimators=500, max_samples=100, bootstrap=True, n_j
obs=-1, random_state=0)

dtree_bag_clf.fit(X_train, y_train)
y_pred = dtree_bag_clf.predict(X_test)
```

In [105]:

```
dtree_bag_clf.fit(X_train, y_train)

# train and test scores
print('Train score: %.2f'%dtree_bag_clf.score(X_train, y_train))
print('Test score: %.2f'%dtree_bag_clf.score(X_test, y_test))
```

```
Train score: 0.91
Test score: 0.91
```

In [106]:

```
print(confusion_matrix(y_test, dtree_bag_clf.predict(X_test) ))
```

```
from sklearn.metrics import classification_report
print(classification_report(y_train, dtree_bag_clf.predict(X_train)))
```

```
[[7128  180]
 [ 557  368]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95     29218
           1       0.67      0.39      0.49      3713

    accuracy                           0.91     32931
   macro avg       0.80      0.68      0.72     32931
weighted avg       0.90      0.91      0.90     32931
```

Random Forest Classifier

GridSearch

In [107]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {'n_estimators': [200, 300, 400, 500],
              'max_depth': np.arange(1, 10)}

rf_gridsearch = GridSearchCV(RandomForestClassifier(random_state=0), param_grid, cv=10,
return_train_score=True)
rf_gridsearch.fit(X_train, y_train)
print("Best parameters for RandomForest Clf: {}".format(rf_gridsearch.best_params_))
print("Best cross-validation score: {:.2f}".format(rf_gridsearch.best_score_))
```

```
Best parameters for RandomForest Clf: {'max_depth': 9, 'n_estimators': 500}
Best cross-validation score: 0.91
```

Random Forest Classifier

In [108]:

```
rf_clf = RandomForestClassifier(n_estimators=400, max_depth = 9, bootstrap=True, n_jobs=-1, random_
state=0)
rf_clf.fit(X_train, y_train)

pred_rf = rf_clf.predict(X_test)

#train and test scores
print('Train score: {:.2f}'.format(rf_clf.score(X_train, y_train)))
print('Test score: {:.2f}'.format(rf_clf.score(X_test, y_test)))
```

```
Train score: 0.92
Test score: 0.91
```

In [109]:

```
print(confusion_matrix(y_test, rf_clf.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, rf_clf.predict(X_train)))
```

```
[[7189  119]
 [ 641  284]]
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     29218
           1       0.86      0.39      0.54      3713

    accuracy                           0.92     32931
   macro avg       0.89      0.69      0.75     32931
```

```
   macro avg       0.89      0.69      0.75     32931
weighted avg       0.92      0.92      0.91     32931
```

Pasting

Decision Tree Classifier

In [110]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# pasting: bootstrap = False

dtree_clf = DecisionTreeClassifier(criterion = 'entropy', random_state=0)
dtree_bag_clf = BaggingClassifier(dtree_clf, n_estimators=500, max_samples=100, bootstrap=False, ra
ndom_state=0)

dtree_bag_clf.fit(X_train, y_train)
y_pred = dtree_bag_clf.predict(X_test)

from  sklearn.metrics import accuracy_score

# train and test scores
print('Train score: %.2f'%dtree_bag_clf.score(X_train, y_train))
print('Test score: %.2f'%dtree_bag_clf.score(X_test, y_test))
```

```
Train score: 0.91
Test score: 0.91
```

In [111]:

```
print(confusion_matrix(y_test, dtree_bag_clf.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, dtree_bag_clf.predict(X_train)))
```

```
[[7172  136]
 [ 621  304]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     29218
           1       0.70      0.32      0.44      3713

    accuracy                           0.91     32931
   macro avg       0.81      0.65      0.69     32931
weighted avg       0.89      0.91      0.89     32931
```

SVC Classifier

In [112]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# pasting: bootstrap = False

svc_clf = SVC(C = 0.1, gamma = 0.01, probability = True, random_state= 0)
svc_bag_clf = BaggingClassifier(svc_clf, n_estimators=500, max_samples=100, bootstrap=False, random
_state=0)

svc_bag_clf.fit(X_train, y_train)
y_pred = svc_bag_clf.predict(X_test)

from  sklearn.metrics import accuracy_score

#train and test scores
print('Train score: %.2f'%svc_bag_clf.score(X_train, y_train))
```

```
print('Train score: %.2f'%svc_bag_clf.score(X_train, y_train))
print('Test score: %.2f'%svc_bag_clf.score(X_test, y_test))
```

```
Train score: 0.89
Test score: 0.89
```

In [113]:

```
print(confusion_matrix(y_test, svc_bag_clf.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, svc_bag_clf.predict(X_train)))
```

```
[[7289   19]
 [ 875   50]]
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     29218
           1       0.86      0.06      0.11      3713

    accuracy                           0.89     32931
   macro avg       0.87      0.53      0.53     32931
weighted avg       0.89      0.89      0.85     32931
```

ADA Boost Classifier

Decision Tree

In [114]:

```
from sklearn.ensemble import AdaBoostClassifier

dtree_ada_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=200,
                             algorithm="SAMME.R", learning_rate=0.5, random_state=0)

dtree_ada_clf.fit(X_train, y_train)
predictions = dtree_ada_clf.predict(X_test)

#train and test scores
print('Train score: %.2f'%dtree_ada_clf.score(X_train, y_train))
print('Test score: %.2f'%dtree_ada_clf.score(X_test, y_test))
```

```
Train score: 0.91
Test score: 0.91
```

In [115]:

```
confusion_matrix(y_test, predictions)
```

Out[115]:

```
array([[7108,  200],
       [ 560,  365]], dtype=int64)
```

In [116]:

```
print(confusion_matrix(y_test, dtree_ada_clf.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, dtree_ada_clf.predict(X_train)))
```

```
[[7108  200]
 [ 560  365]]
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     29218
           1       0.66      0.40      0.50      3713
```

```
    accuracy                           0.91     32931
   macro avg       0.79      0.69      0.72     32931
weighted avg       0.90      0.91      0.90     32931
```

Logistic Regression

In [117]:

```
from sklearn.ensemble import AdaBoostClassifier

log_reg_ada_clf = AdaBoostClassifier(LogisticRegression(solver='liblinear'), n_estimators=500,
                            algorithm="SAMME.R", learning_rate=0.1, random_state=0)

log_reg_ada_clf.fit(X_train, y_train)
predictions = log_reg_ada_clf.predict(X_test)

#train and test scores
print('Train score: %.2f'%log_reg_ada_clf.score(X_train, y_train))
print('Test score: %.2f'%log_reg_ada_clf.score(X_test, y_test))
```

```
Train score: 0.91
Test score: 0.91
```

In [118]:

```
print(confusion_matrix(y_test, log_reg_ada_clf.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, log_reg_ada_clf.predict(X_train)))
```

```
[[7194  114]
 [ 663  262]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     29218
           1       0.70      0.28      0.40      3713

    accuracy                           0.91     32931
   macro avg       0.81      0.63      0.68     32931
weighted avg       0.89      0.91      0.89     32931
```

Gradient Boosting Classifier

In [119]:

```
from  sklearn.ensemble import GradientBoostingClassifier

gbrt = GradientBoostingClassifier(random_state=0, max_depth=5, learning_rate=0.01)
gbrt.fit(X_train, y_train)

#train and test scores
print("Accuracy on training set: {:.3f}".format(gbrt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(X_test, y_test)))
```

```
Accuracy on training set: 0.904
Accuracy on test set: 0.901
```

In [120]:

```
print(confusion_matrix(y_test, gbrt.predict(X_test)))
from sklearn.metrics import classification_report
print(classification_report(y_train, gbrt.predict(X_train)))
```

```
[[7276   32]
 [ 779  146]]
              precision    recall  f1-score   support
```

```
           0       0.90      1.00      0.95     29218
           1       0.87      0.17      0.28      3713

    accuracy                           0.90     32931
   macro avg       0.89      0.58      0.62     32931
weighted avg       0.90      0.90      0.87     32931
```

PCA

In [121]:

```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X_train)

cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

In [122]:

```
d
```

Out[122]:

```
29
```

In [123]:

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

In [124]:

```
pca.n_components_
```

Out[124]:

```
29
```

In [125]:

```
np.sum(pca.explained_variance_ratio_)
```

Out[125]:

```
0.9538315288981813
```

In [126]:

```
pca = PCA(n_components = 29)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

In [127]:

```
X_reduced_pca = X_reduced
```

In [128]:

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100

inc_pca = IncrementalPCA(n_components=29)
```

```
inc_pca = IncrementalPCA(n_components=29)
for X_batch in np.array_split(X_train, n_batches):
    print(".", end="")
    inc_pca.partial_fit(X_batch)

X_train_reduced = inc_pca.transform(X_train)
```

```
....................................................................................................
```

In [129]:

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100

inc_pca = IncrementalPCA(n_components=29)
for X_batch in np.array_split(X_test, n_batches):
    print(".", end="")
    inc_pca.partial_fit(X_batch)

X_test_reduced = inc_pca.transform(X_test)
```

```
....................................................................................................
```

In [130]:

```
X_train_reduced.shape
```

Out[130]:

```
(32931, 29)
```

In [131]:

```
X_test_reduced.shape
```

Out[131]:

```
(8233, 29)
```

Models on PCA data

Logistic Regression

In [132]:

```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
log_model.fit(X_train_reduced,y_train)
log_pred = log_model.predict(X_test_reduced)
```

In [133]:

```
print(confusion_matrix(y_test, log_pred))
from sklearn.metrics import classification_report
print(classification_report(y_train, log_model.predict(X_train_reduced)))
```

```
[[7086  222]
 [ 711  214]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     29218
           1       0.65      0.36      0.46      3713

    accuracy                           0.91     32931
   macro avg       0.79      0.67      0.70     32931
weighted avg       0.89      0.91      0.89     32931
```

In [134]:

```
lr_score_train = log_model.score(X_train_reduced,y_train)
lr_score_train
```

Out[134]:

```
0.9056208435820352
```

In [135]:

```
lr_score_test = log_model.score(X_test_reduced,y_test)
lr_score_test
```

Out[135]:

```
0.886675573909875
```

KNN Classifier

In [136]:

```
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier


neighbors = np.arange(0,25)

cv_scores = []
```

In [137]:

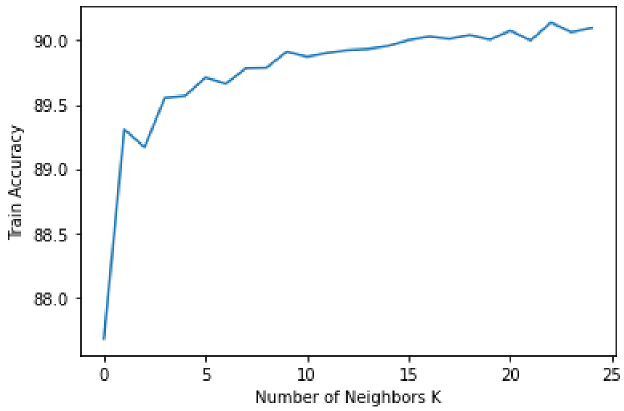
```
# To determine best k-value
for k in neighbors:
    k_val = k+1
    knn_clf = KNeighborsClassifier(n_neighbors = k_val, weights='uniform', p=2, metric='euclidean')
    k_fold = model_selection.KFold(n_splits=10, random_state=123)
    cross_val_scores = model_selection.cross_val_score(knn_clf, X_train_reduced, y_train, cv=k_fold
, scoring='accuracy')
    cv_scores.append(cross_val_scores.mean()*100)
    print("k=%d %0.2f (+/- %0.2f)" % (k_val, cross_val_scores.mean()*100, cross_val_scores.std()*10
0))

optimal_kval = neighbors[cv_scores.index(max(cv_scores))]
print ("The optimal number of neighbors is %d with %0.1f%%" % (optimal_kval,
cv_scores[optimal_kval]))

plt.plot(neighbors, cv_scores)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Train Accuracy')
plt.show()
```

```
k=1 87.67 (+/- 0.61)
k=2 89.31 (+/- 0.62)
k=3 89.17 (+/- 0.43)
k=4 89.55 (+/- 0.53)
k=5 89.57 (+/- 0.44)
k=6 89.71 (+/- 0.47)
k=7 89.66 (+/- 0.35)
k=8 89.78 (+/- 0.45)
k=9 89.79 (+/- 0.43)
k=10 89.91 (+/- 0.37)
k=11 89.87 (+/- 0.35)
k=12 89.90 (+/- 0.45)
k=13 89.92 (+/- 0.45)
k=14 89.93 (+/- 0.46)
k=15 89.96 (+/- 0.40)
k=16 90.00 (+/- 0.46)
k=17 90.03 (+/- 0.39)
```

```
k=17 90.05 (+/- 0.39)
k=18 90.01 (+/- 0.46)
k=19 90.04 (+/- 0.46)
k=20 90.01 (+/- 0.53)
k=21 90.08 (+/- 0.48)
k=22 90.00 (+/- 0.50)
k=23 90.14 (+/- 0.46)
k=24 90.06 (+/- 0.44)
k=25 90.10 (+/- 0.47)
The optimal number of neighbors is 22 with 90.1%
```



In [138]:

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=24)
knn.fit(X_train_reduced, y_train)
knn_pred = knn.predict(X_test_reduced)
```

In [139]:

```python
# for train data
from sklearn.metrics import classification_report
print(classification_report(y_train, knn.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     29218
           1       0.71      0.25      0.37      3713

    accuracy                           0.90     32931
   macro avg       0.81      0.62      0.66     32931
weighted avg       0.89      0.90      0.88     32931
```

In [140]:

```python
# for test data
confusion_matrix(y_test, knn.predict(X_test_reduced))
```

Out[140]:

```
array([[7249,   59],
       [ 875,   50]], dtype=int64)
```

In [141]:

```python
print(classification_report(y_test, knn.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      7308
           1       0.46      0.05      0.10       925

    accuracy                           0.89      8233
   macro avg       0.68      0.52      0.52      8233
```

```
weighted avg       0.84      0.89      0.84      8233
```

In [142]:

```
knn_score_test = knn.score(X_test_reduced,y_test)
knn_score_test
```

Out[142]:

```
0.88655411150249
```

Linear SVM

In [143]:

```
from sklearn.svm import LinearSVC

svm = LinearSVC()
svm.fit(X_train_reduced,y_train)
svc_pred = svm.predict(X_test_reduced)
```

In [144]:

```
# for train set
from sklearn.metrics import classification_report
print(classification_report(y_train, svm.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     29218
           1       0.67      0.31      0.42      3713

    accuracy                           0.90     32931
   macro avg       0.79      0.64      0.68     32931
weighted avg       0.89      0.90      0.89     32931
```

In [145]:

```
#for test set
from sklearn.metrics import classification_report
print(classification_report(y_test, svm.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7308
           1       0.48      0.17      0.25       925

    accuracy                           0.89      8233
   macro avg       0.69      0.57      0.59      8233
weighted avg       0.86      0.89      0.86      8233
```

In [146]:

```
svm_score_train = svm.score(X_train_reduced,y_train)
svm_score_train
```

Out[146]:

```
0.904588381767939
```

In [147]:

```
svm_score_test = svm.score(X_test_reduced,y_test)
svm_score_test
```

Out[147]:

```
0.8860682618729503
```

Decision Tree Classifier

In [148]:

```
d_tree = DecisionTreeClassifier(criterion='entropy', random_state=0)
d_tree.fit(X_train_reduced, y_train)
d_tree_pred = d_tree.predict(X_test_reduced)
```

In [149]:

```
print(classification_report(y_test, d_tree.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.89      7308
           1       0.23      0.28      0.26       925

    accuracy                           0.81      8233
   macro avg       0.57      0.58      0.57      8233
weighted avg       0.83      0.81      0.82      8233
```

In [150]:

```
print(classification_report(y_train, d_tree.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     29218
           1       1.00      1.00      1.00      3713

    accuracy                           1.00     32931
   macro avg       1.00      1.00      1.00     32931
weighted avg       1.00      1.00      1.00     32931
```

In [151]:

```
d_tree_score_train = d_tree.score(X_train_reduced,y_train)
d_tree_score_train
```

Out[151]:

```
0.9999696334760559
```

In [152]:

```
d_tree_score_test = d_tree.score(X_test_reduced,y_test)
d_tree_score_test
```

Out[152]:

```
0.814283979108466
```

Kernalized SVM(linear, rbf, poly)

In [153]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

tuned_parameters = [{'kernel': ['rbf'], 'gamma': [0.1],
```

```
                    'C': [1]},
                   {'kernel': ['linear'], 'C': [1]},
                  {'kernel' : ['poly'], 'degree':[3], 'C':[10] }]
```

In [154]:

```
clf = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='precision')
clf.fit(X_train, y_train)

print(clf.cv_results_)
```

```
{'mean_fit_time': array([91.10583048, 70.27198591, 80.00121274]), 'std_fit_time':
array([1.04248985, 7.38563372, 1.80327034]), 'mean_score_time': array([5.95953856, 2.66312752,
2.90309567]), 'std_score_time': array([0.01897421, 0.03878049, 0.03692636]), 'param_C':
masked_array(data=[1, 1, 10],
             mask=[False, False, False],
       fill_value='?',
            dtype=object), 'param_gamma': masked_array(data=[0.1, --, --],
             mask=[False,  True,  True],
       fill_value='?',
            dtype=object), 'param_kernel': masked_array(data=['rbf', 'linear', 'poly'],
             mask=[False, False, False],
       fill_value='?',
            dtype=object), 'param_degree': masked_array(data=[--, --, 3],
             mask=[ True,  True, False],
       fill_value='?',
            dtype=object), 'params': [{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}, {'C': 1, 'kernel': '
linear'}, {'C': 10, 'degree': 3, 'kernel': 'poly'}], 'split0_test_score': array([0.67412141,
0.63099631, 0.60408163]), 'split1_test_score': array([0.63467492, 0.62939297, 0.53682171]),
'split2_test_score': array([0.62170088, 0.65116279, 0.56097561]), 'split3_test_score':
array([0.63276836, 0.64312268, 0.59578544]), 'split4_test_score': array([0.59744409, 0.62666667, 0
.5462963 ]), 'mean_test_score': array([0.63214193, 0.63626828, 0.56879214]), 'std_test_score':
array([0.02482757, 0.00933899, 0.02669553]), 'rank_test_score': array([2, 1, 3])}
```

In [155]:

```
print('The best model is: ', clf.best_params_)
print('This model produces a mean cross-validated score (precision) of', clf.best_score_)
```

```
The best model is:  {'C': 1, 'kernel': 'linear'}
This model produces a mean cross-validated score (precision) of 0.6362682830306745
```

In [156]:

```
svm_ker_lin = SVC(kernel='linear', C=1)
svm_ker_rbf = SVC(kernel='rbf', gamma=0.1, C=1)
svm_ker_poly = SVC(kernel='poly', degree=3, C=10)
```

In [157]:

```
svm_ker_lin.fit(X_train_reduced, y_train)
svm_ker_rbf.fit(X_train_reduced, y_train)
svm_ker_poly.fit(X_train_reduced, y_train)

ker_lin_pred = svm.predict(X_test_reduced)
ker_rbf_pred = svm.predict(X_test_reduced)
ker_poly_pred = svm.predict(X_test_reduced)
```

In [158]:

```
print(classification_report(y_train, svm_ker_lin.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     29218
           1       0.65      0.29      0.41      3713

    accuracy                           0.90     32931
   macro avg       0.79      0.64      0.68     32931
weighted avg       0.89      0.90      0.89     32931
```

In [159]:

```
print(classification_report(y_train, svm_ker_rbf.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.97     29218
           1       0.89      0.52      0.66      3713

    accuracy                           0.94     32931
   macro avg       0.92      0.76      0.81     32931
weighted avg       0.94      0.94      0.93     32931
```

In [160]:

```
print(classification_report(y_train, svm_ker_poly.predict(X_train_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     29218
           1       0.80      0.50      0.62      3713

    accuracy                           0.93     32931
   macro avg       0.87      0.74      0.79     32931
weighted avg       0.92      0.93      0.92     32931
```

In [161]:

```
print(classification_report(y_test, svm_ker_poly.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      7308
           1       0.30      0.32      0.31       925

    accuracy                           0.84      8233
   macro avg       0.61      0.61      0.61      8233
weighted avg       0.84      0.84      0.84      8233
```

In [162]:

```
print(classification_report(y_test, svm_ker_lin.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      7308
           1       0.53      0.15      0.24       925

    accuracy                           0.89      8233
   macro avg       0.72      0.57      0.59      8233
weighted avg       0.86      0.89      0.86      8233
```

In [163]:

```
print(classification_report(y_test, svm_ker_rbf.predict(X_test_reduced)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      7308
           1       0.31      0.02      0.04       925

    accuracy                           0.88      8233
   macro avg       0.60      0.51      0.49      8233
```

```
weighted avg       0.82      0.88      0.84      8233
```

In [164]:

```
# Since linear kernel is our best model, we will consider it's train and test scores.
#test score
svm_ker_lin_score_test = svm_ker_lin.score(X_test_reduced,y_test)
svm_ker_lin_score_test
```

Out[164]:

```
0.8895906716871128
```

In [165]:

```
#train score
svm_ker_lin_score_train = svm_ker_lin.score(X_train_reduced,y_train)
svm_ker_lin_score_train
```

Out[165]:

```
0.9028574899031307
```

Results from Project 1 for all models :

Train scores for our models are as follows :

Logistic Regression : 0.9092951929792596

KNN Classification : 0.9031915216665148

Linear SVM : 0.9085056633567156

Kernalized SVM : 0.8997904709847864

Decision Tree Classifier : 0.9999696334760559

Test scores for our models are as follows :

Logistic Regression : 0.9086602696465444

KNN Classification : 0.9001579011296004

Linear SVM : 0.9069597959431556

Kernalized SVM : 0.9086602696465444

Decision Tree Classifier : 0.8933560063160452

As we can see from our train and test scores from Project 1 and our train and test scores when using our PCA reduced dataset, the scores have dropped. Hence, we can say that the dimensionally reduced dataset results in a poorer score compared to the original dataset. Although, PCA did help in saving computational time by reducing the features to 29 from 38.

In [166]:

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

# create model
clf_model = Sequential()
clf_model.add(Dense(12, input_dim=45, activation='relu'))
clf_model.add(Dense(8, activation='relu'))
```

```
clf_model.add(Dense(1, activation='sigmoid'))

# Compile model
clf_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
return model

# Fit the model
clf_model.fit(X_train, y_train, epochs=150, batch_size=10)

# evaluate the model
clf_model_scores = clf_model.evaluate(X_test, y_test)
print("\n%s: %.2f%%" % (clf_model.metrics_names[1], scores[1]*100))
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-166-2a2fa6886b37> in <module>
      1 import numpy
----> 2 from keras.models import Sequential
      3 from keras.layers import Dense
      4 from keras.wrappers.scikit_learn import KerasClassifier
      5 

ModuleNotFoundError: No module named 'keras'
```

In [ ]:

```
y_predict = clf_model.predict(X_test)
y_predict
```

In [ ]: