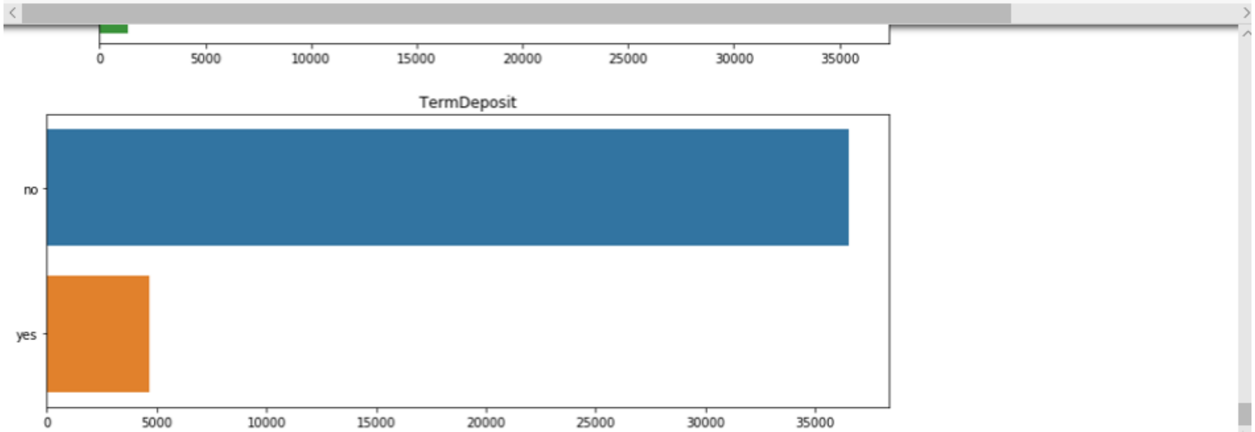
Data Wrangling

Exploratory Analysis:

We first start the exploratory analysis of the missing or Unknown values of the categorical variables which are use for predicting the outcome Term Deposit. After loading the data we rename the columns and check the categorical variables (Job, Marital, Education, HasCredit, HouseLoan, PersonalLoan, ContactType, LastContMonth, LastContDay, PrevOut, TermDeposit) for missing values.

```
categorical_variables = ['Job', 'Education', 'HasCredit', 'HouseLoan', 'PersonalLoan', 'ContactType', 'LastContMonth', 'LastContD
for col in categorical_variables:
    plt.figure(figsize=(10,4))
    sns.barplot(df[col].value_counts().values, df[col].value_counts().index)
    plt.title(col)
    plt.tight_layout()
    plt.show()
```



There are unknown values for many variables in the Data set. One way to handle is to discard the row but that would lead to reduction of data set which wouldn't serve the purpose of building accurate and realistic prediction model. Another way is to infer the value from other variables, however it doesn't guarantee that all the missing values will be address but majority of them will be cleaned up for analysis.

Variables with unknown values are : Education, Job, HasCredit, HouseLoan, PersonalLoan and Marital. However the Marital status has few unknown values, significant ones are Education, Job, HouseLoan and PersonalLoan. We will try to see the pattern for these missing values.

We will write a function which will return the Variable 1 groupby Variable 2 unique value counts as DataFrame.
 We will use this function to check the missing/unknown values and see if can draw ay intuitions in filling the unknown values.

```python
def var_test(df,f1,f2):
    var1 = list(df[f1].unique())
    var2 = list(df[f2].unique())
    dataframes = []
    for e in var2:
        dfv2 = df[df[f2]==e]
        dfv1 = dfv2.groupby(f1).count()[f2]
        dataframes.append(dfv1)
    xx=pd.concat(dataframes, axis=1)
    xx.columns=var2
    xx=xx.fillna(0)
    return dfv2
```

```python
var_test(df,'Job','Education')
```

	basic.4y	high.school	basic.6y	basic.9y	professional.course	unknown	university.degree	illiterate
admin.	77	3329	151	499	363	249	5753	1.0
blue-collar	2318	878	1426	3623	453	454	94	8.0
entrepreneur	137	234	71	210	135	57	610	2.0
housemaid	474	174	77	94	59	42	139	1.0
management	100	298	85	166	89	123	2063	0.0
retired	597	276	75	145	241	98	285	3.0
self-employed	93	118	25	220	168	29	765	3.0
services	132	2682	226	388	218	150	173	0.0
student	26	357	13	99	43	167	170	0.0
technician	58	873	87	384	3320	212	1809	0.0
unemployed	112	259	34	186	142	19	262	0.0
unknown	52	37	22	31	12	131	45	0.0

Inferring Education from Jobs: From the above table it can be seen that people with management will usually have a university degree, so we can replace the 'unknown' with 'university degree'. Similarly job with 'services' education as 'high.school', job with 'housemaid' education as 'basic.4y'.

Similarly we can also infer the jobs from education where 'Education' = 'basic.4y' or 'basic.6y' or 'basic.9y' with job as 'blue-collar', if Education is 'professional.course' the job = 'technician'.

It would also make sense to replace the unknown values for job where age > 60 as 'retired'.

```
df.loc[(df.Age>60) & (df.Job=='unknown'),'Job'] = 'retired'
df.loc[(df.Education=='unknown') & (df.Job=='management'), 'Education'] = 'university.degree'
df.loc[(df.Education=='unknown') & (df.Job=='services'), 'Education'] = 'high.school'
df.loc[(df.Education=='unknown') & (df.Job=='housemaid'), 'Education'] = 'basic.4y'
df.loc[(df.Job=='unknown') & (df.Education=='basic.4y'), 'Job'] = 'blue-collar'
df.loc[(df.Job=='unknown') & (df.Education=='basic.6y'), 'Job'] = 'blue-collar'
df.loc[(df.Job=='unknown') & (df.Education=='basic.9y'), 'Job'] = 'blue-collar'
df.loc[(df.Job=='unknown') & (df.Education=='professional.course'), 'Job'] = 'technician'
```

```
var_dependency(df,'Job','Education')
```

	basic.4y	high.school	basic.6y	basic.9y	professional.course	unknown	university.degree	illiterate
admin.	77.0	3329	151.0	499.0	363.0	249.0	5753	1.0
blue-collar	2366.0	878	1448.0	3654.0	453.0	454.0	94	8.0
entrepreneur	137.0	234	71.0	210.0	135.0	57.0	610	2.0
housemaid	516.0	174	77.0	94.0	59.0	0.0	139	1.0
management	100.0	298	85.0	166.0	89.0	0.0	2186	0.0
retired	601.0	276	75.0	145.0	243.0	112.0	286	3.0
self-employed	93.0	118	25.0	220.0	168.0	29.0	765	3.0
services	132.0	2832	226.0	388.0	218.0	0.0	173	0.0
student	26.0	357	13.0	99.0	43.0	167.0	170	0.0
technician	58.0	873	87.0	384.0	3330.0	212.0	1809	0.0
unemployed	112.0	259	34.0	186.0	142.0	19.0	262	0.0
unknown	0.0	37	0.0	0.0	0.0	117.0	44	0.0

Numerical Variables:

Let see the summary of data in order to understand the numerical vairables.

```
numerical_variables = ['Age', 'Campaign', 'PreviousDay', 'PrevContNum', 'EmpVarRate', 'ConsumerPriceIdx', 'ConsConfIdx', 'Euribor
df[numerical_variables].describe()
```

	Age	Campaign	PreviousDay	PrevContNum	EmpVarRate	ConsumerPriceIdx	ConsConfIdx	Euribor	Employeeno
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	10.42125	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528
min	17.00000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

Missing Values: From the available dataset description, missing values or NaNs are encoded as '999'. From the above screen it is clear that only PreviousDay has majority of missing values.
To deal with this variable, we will remove numerical variable PreviousDay and replace with additional categorical variables as following categories: pdays_missing (0 for contacted before and 1 for not previously contacted) , pdays_less_5, pdays_betw_5_15 and pdays_greater_15.

```
df['pdays_missing'] = 0
df['pdays_less_5'] = 0
df['pdays_betw_5_15'] = 0
df['pdays_greater_15'] = 0
df['pdays_missing'][df['PreviousDay']==999] = 1
df['pdays_less_5'][df['PreviousDay']<5] = 1
df['pdays_betw_5_15'][(df['PreviousDay']>=5) & (df['PreviousDay']<=15)] = 1
df['pdays_greater_15'][(df['PreviousDay']>15) & (df['PreviousDay'] < 999)] = 1
df_dropped_pdays = df.drop('PreviousDay', axis=1)
```