Bank_marketing_2nd_improvement

April 10, 2019

0.1 Definition of Input Variables

- age - Age of the client- (numeric)
- job - Client's occupation - (categorical) (admin, bluecollar, entrepreneur, housemaid, management, retired, selfemployed, services, student, technician, unemployed, unknown)
- marital - Client's marital status - (categorical) (divorced, married, single, unknown, note: divorced means divorced or widowed)
- education - Client's education level - (categorical) (basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown)
- default - Indicates if the client has credit in default - (categorical) (no, yes, unknown)
- housing - Does the client have a housing loan? - (categorical) (no, yes, unknown)
- loan - Does the client have a personal loan? - (categorical) (no, yes, unknown')
- contact - Type of communication contact - (categorical) (cellular, telephone)
- month - Month of last contact with client - (categorical) (January - December)
- day_of_week - Day of last contact with client - (categorical) (Monday - Friday)
- duration - Duration of last contact with client, in seconds - (numeric)
- campaign - Number of client contacts during this campaign - (numeric) (includes last contact)
- pdays - Number of days from last contacted from a previous campaign - (numeric) (999 means client was not previously contacted)
- previous - Number of client contacts performed before this campaign - (numeric)
- poutcome - Previous marketing campaign outcome - (categorical) (failure, nonexistent , success)
- emp.var.rate - Quarterly employment variation rate - (numeric)
- cons.price.idx - Monthly consumer price index - (numeric)
- cons.conf.idx - Monthly consumer confidence index - (numeric)

- euribor3m - Daily euribor 3 month rate - (numeric)
- nr.employed - Quarterly number of employees - (numeric)
- Output variable (desired target) - Term Deposit - subscription verified (binary: 'yes','no')

0.2 Imports

```
In [152]: #Data Storage and Manipulation Libraries
          import pandas as pd
          import numpy as np

          # !pip3 install sklearn

          import sklearn
          from sklearn.model_selection import train_test_split
          from sklearn.model_selection import ShuffleSplit
          from sklearn.preprocessing import LabelEncoder
          from sklearn.preprocessing import OneHotEncoder
          from sklearn.feature_extraction import DictVectorizer
          from imblearn.over_sampling import SMOTE

          #models
          from sklearn.linear_model import LogisticRegression
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.svm import SVC
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoos
          from sklearn.naive_bayes import GaussianNB
          from sklearn.discriminant_analysis import LinearDiscriminantAnalysis


          #metrics
          from sklearn import metrics as m
          from sklearn.metrics import roc_auc_score
          from sklearn.metrics import roc_curve
          from sklearn.metrics import confusion_matrix
          from sklearn.metrics import classification_report


          #Visualization Libraries
          # !pip3 install seaborn

          import seaborn as sns
          import matplotlib.pyplot as plt
```

0.3 Dataset Path

```
In [153]: path= "/home/nikhil/Downloads/bank-additional/bank-additional/bank-additional-full.cs
```

0.4 Read the Data

```
In [154]: data= pd.read_csv(path, sep=';')
```

0.5 EDA

```
In [155]: data.head()
```

```
Out[155]:    age        job  marital    education  default housing loan    contact  \
          0   56  housemaid  married     basic.4y       no      no   no  telephone
          1   57   services  married  high.school  unknown      no   no  telephone
          2   37   services  married  high.school       no     yes   no  telephone
          3   40     admin.  married     basic.6y       no      no   no  telephone
          4   56   services  married  high.school       no      no  yes  telephone

            month day_of_week  ...  campaign  pdays  previous     poutcome emp.var.rate  \
          0   may         mon  ...         1    999         0  nonexistent          1.1
          1   may         mon  ...         1    999         0  nonexistent          1.1
          2   may         mon  ...         1    999         0  nonexistent          1.1
          3   may         mon  ...         1    999         0  nonexistent          1.1
          4   may         mon  ...         1    999         0  nonexistent          1.1

             cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
          0          93.994          -36.4      4.857       5191.0  no
          1          93.994          -36.4      4.857       5191.0  no
          2          93.994          -36.4      4.857       5191.0  no
          3          93.994          -36.4      4.857       5191.0  no
          4          93.994          -36.4      4.857       5191.0  no

          [5 rows x 21 columns]
```

```
In [156]: print("Shape of the dataset:", data.shape)
```

```
Shape of the dataset: (41188, 21)
```

```
In [157]: print("Columns of the dataset:", data.columns)
```

```
Columns of the dataset: Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loa
       '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')
```

0.6 Datatypes of all features and target variable

```
In [158]: data.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

0.7 Conclusions from the describe function:

- Most customers in this dataset range from 30-50 years

In [159]: data.describe()

Out[159]:

	age	duration	campaign	pdays	previous
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963
std	10.42125	259.279249	2.770014	186.910907	0.494901
min	17.00000	0.000000	1.000000	0.000000	0.000000
25%	32.00000	102.000000	1.000000	999.000000	0.000000
50%	38.00000	180.000000	2.000000	999.000000	0.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000
max	98.00000	4918.000000	56.000000	999.000000	7.000000

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	1.570960	0.578840	4.628198	1.734447	72.251528

```
min       -3.400000       92.201000     -50.800000       0.634000    4963.600000
25%       -1.800000       93.075000     -42.700000       1.344000    5099.100000
50%        1.100000       93.749000     -41.800000       4.857000    5191.000000
75%        1.400000       93.994000     -36.400000       4.961000    5228.100000
max        1.400000       94.767000     -26.900000       5.045000    5228.100000
```

0.8 Check for null values

- No Null Values

In [160]:
```
data.isnull().sum()
```

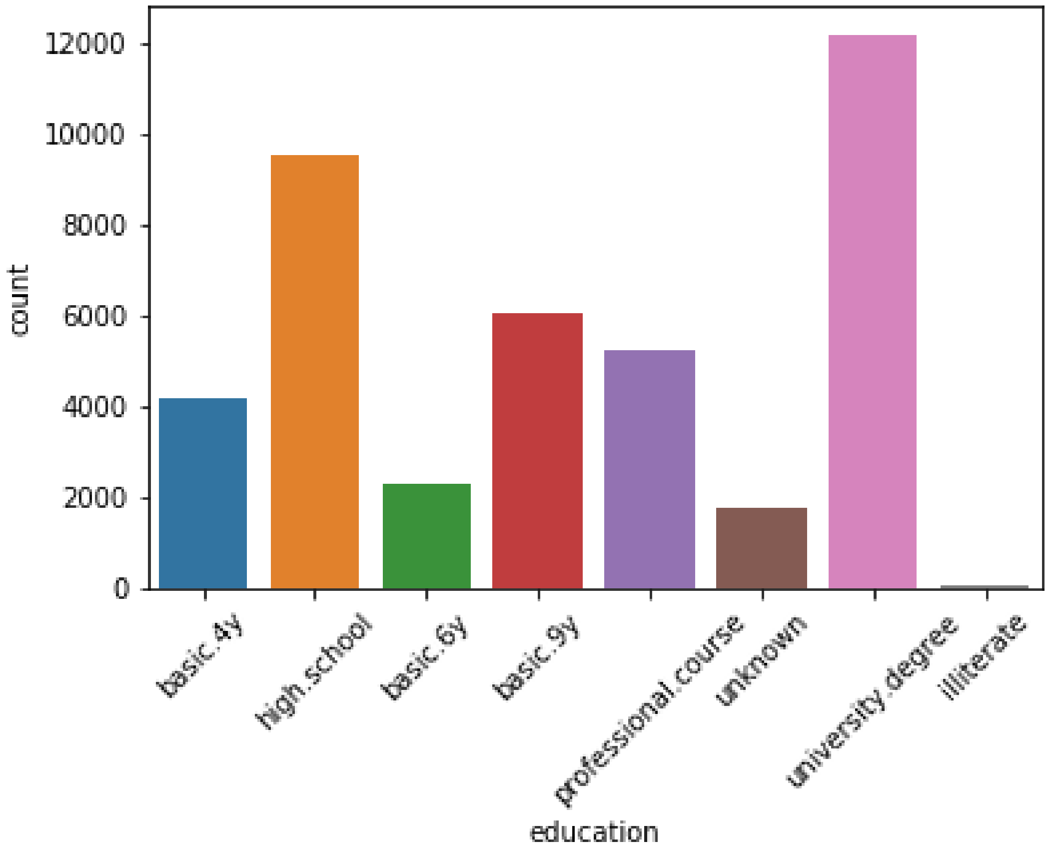
Out[160]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

0.9 Univariate Analysis

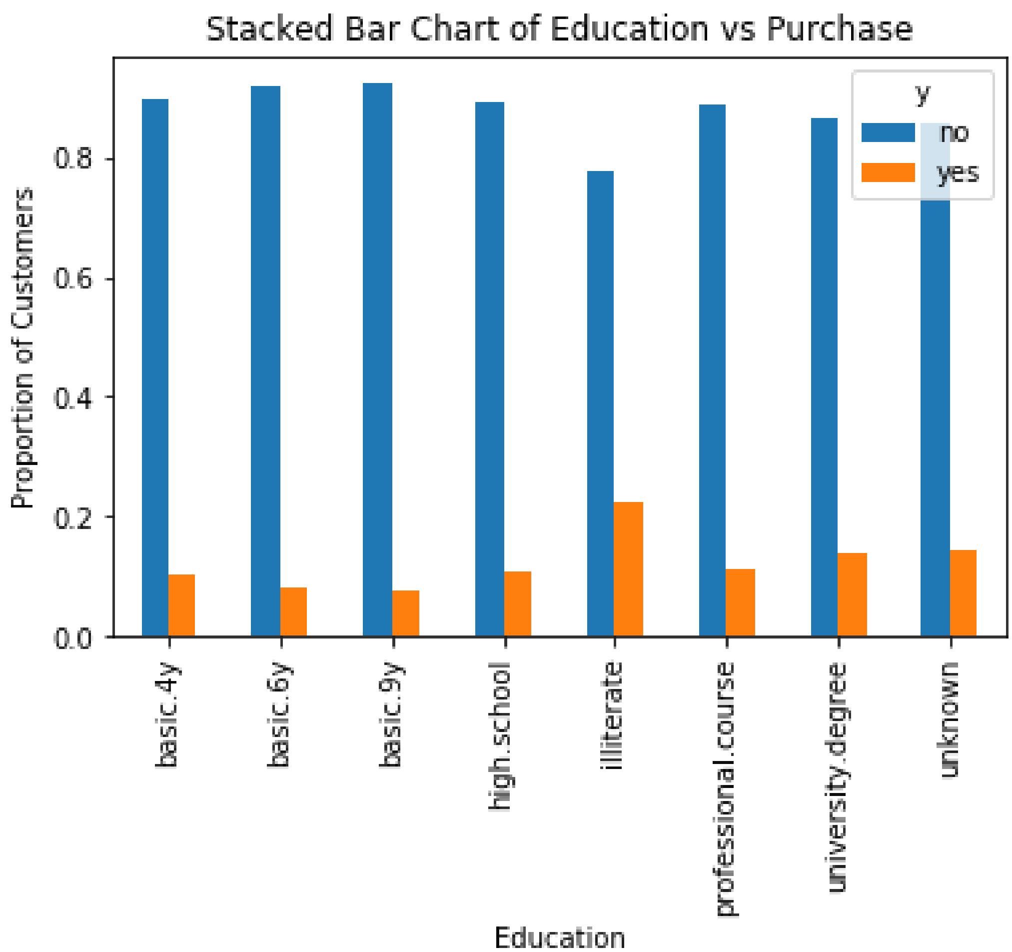
0.10 Education

In [161]:
```
sns.countplot(x='education', data= data)
plt.xticks(rotation=45)
```

Out[161]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]), <a list of 8 Text xticklabel objects>)
```



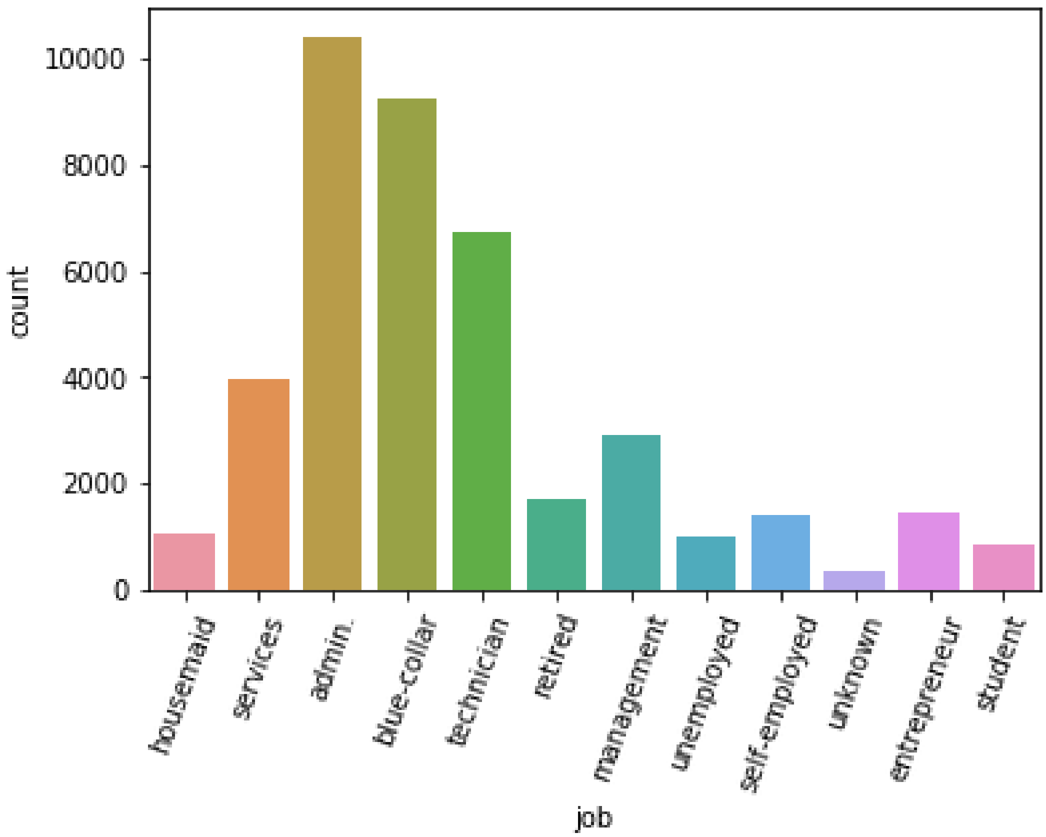

```
In [162]: table=pd.crosstab(data.education,data.y)
          table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=False)
          plt.title('Stacked Bar Chart of Education vs Purchase')
          plt.xlabel('Education')
          plt.ylabel('Proportion of Customers')

Out[162]: Text(0, 0.5, 'Proportion of Customers')
```



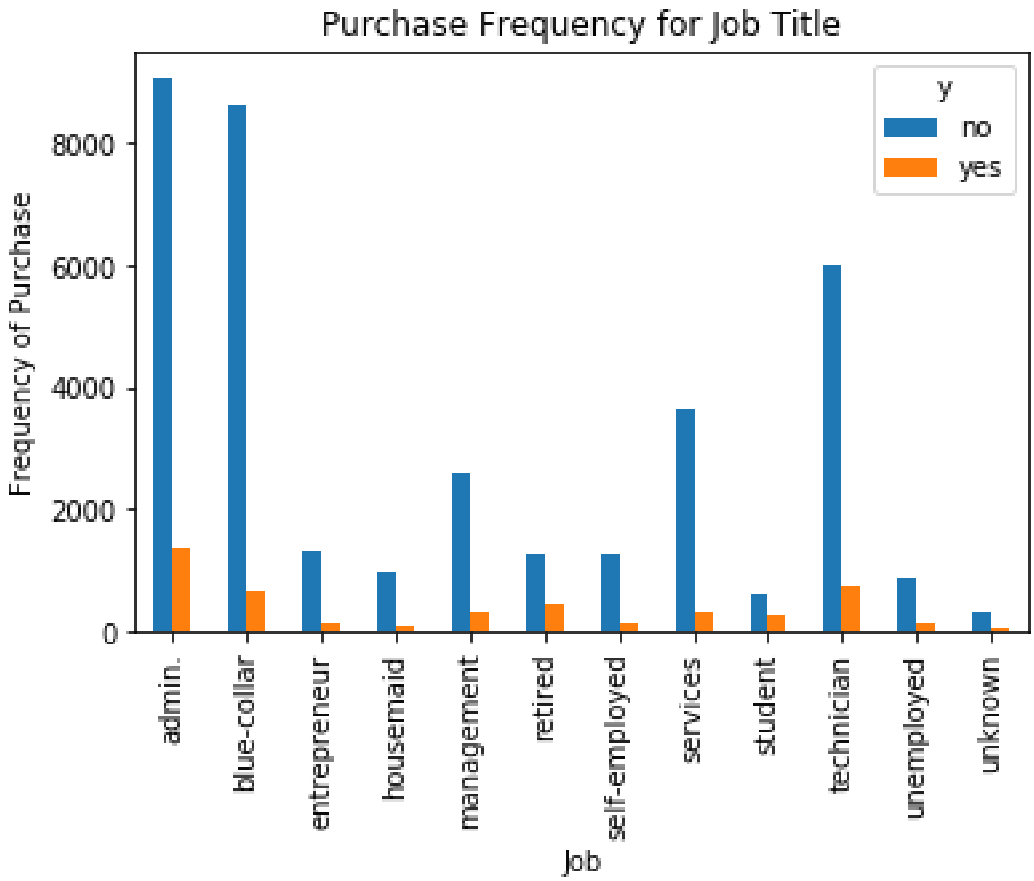
- From this we can infer that Education is an important variable that impacts the target

0.11 Job

```
In [163]: sns.countplot(x='job', data= data)
          plt.xticks(rotation=70)

Out[163]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
           <a list of 12 Text xticklabel objects>)
```



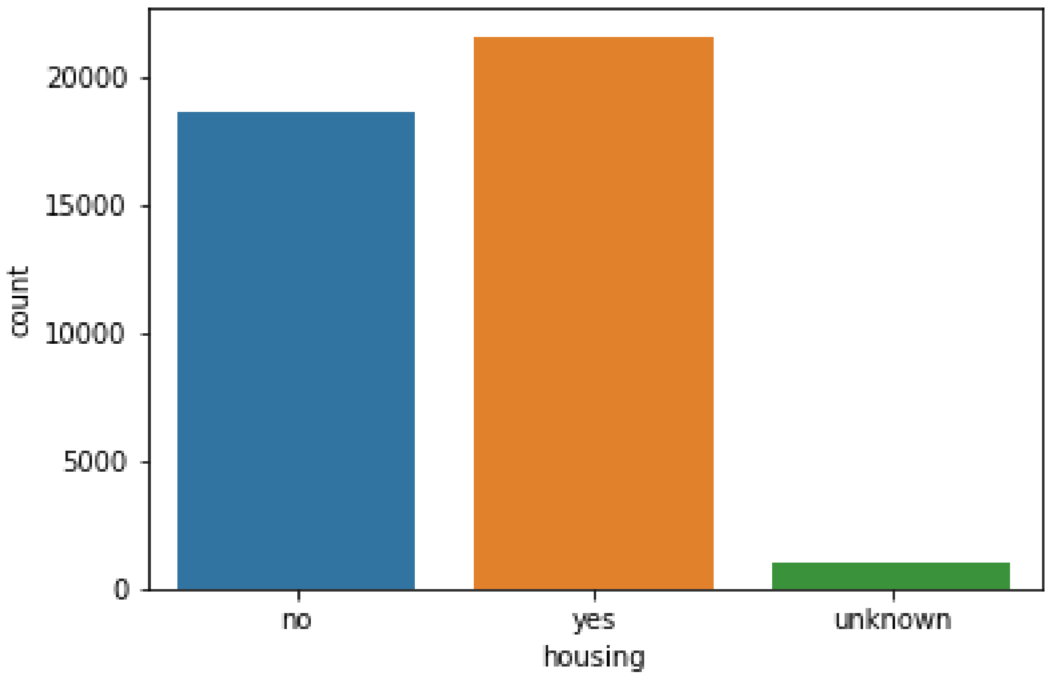

```
In [164]: %matplotlib inline
          pd.crosstab(data.job, data.y).plot(kind='bar')
          plt.title('Purchase Frequency for Job Title')
          plt.xlabel('Job')
          plt.ylabel('Frequency of Purchase')

Out[164]: Text(0, 0.5, 'Frequency of Purchase')
```



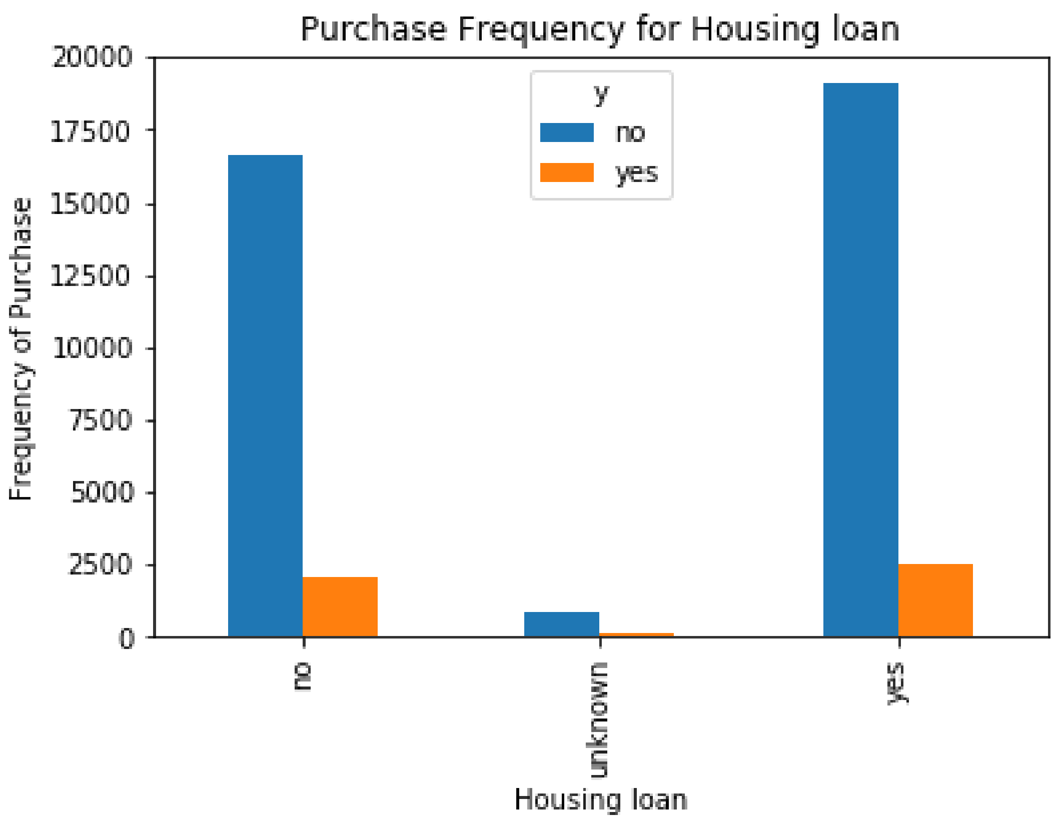
- From this we can infer that Job is an important variable that impacts the target

0.12 Housing

```
In [165]: sns.countplot(x='housing', data= data)

Out[165]: <matplotlib.axes._subplots.AxesSubplot at 0x7f825a01af60>
```




```
In [166]: %matplotlib inline
          pd.crosstab(data.housing, data.y).plot(kind='bar')
          plt.title('Purchase Frequency for Housing loan')
          plt.xlabel('Housing loan')
          plt.ylabel('Frequency of Purchase')

Out[166]: Text(0, 0.5, 'Frequency of Purchase')
```

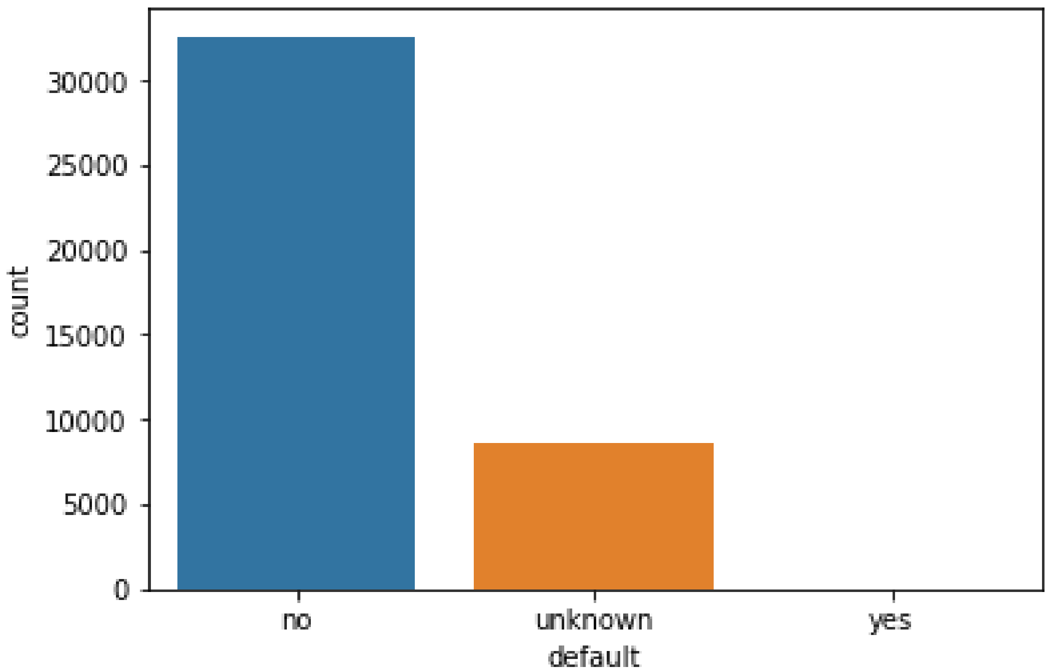


- From this we can infer that having a housing loan is a good variable that impacts the target(although not very apparent)

0.13 Default

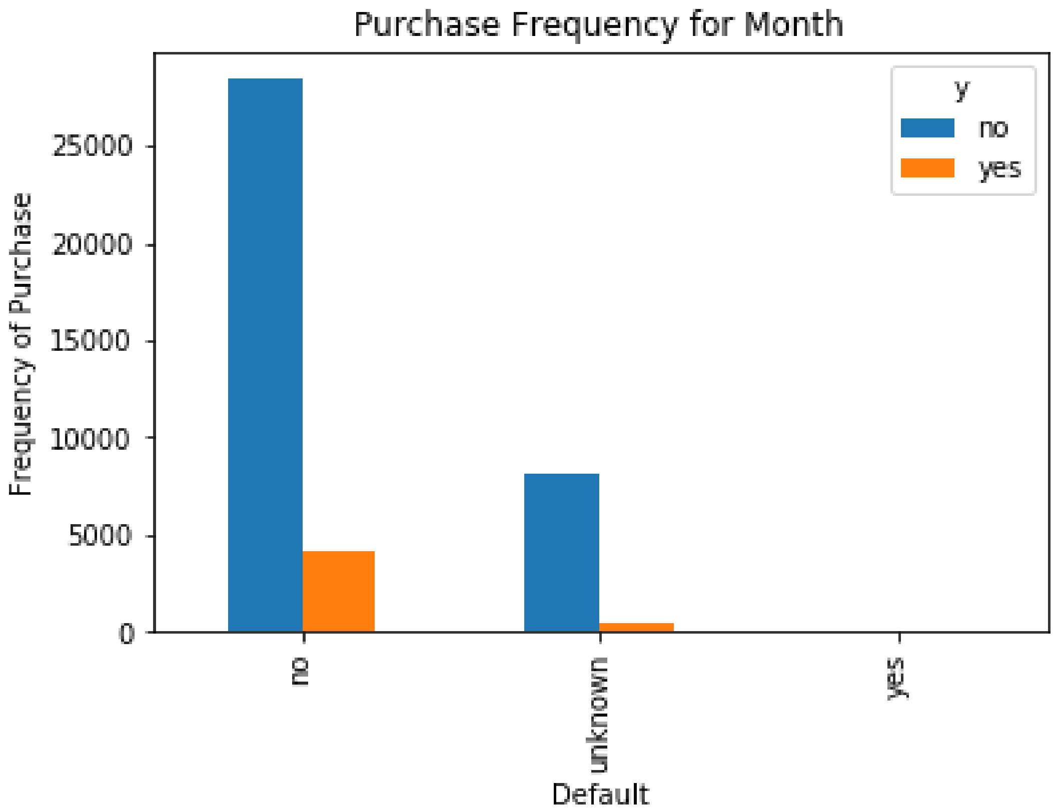
```
In [167]: sns.countplot(x='default', data= data)
```

```
Out[167]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8259ea4320>
```



In [168]:
```
pd.crosstab(data.default,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Month')
plt.xlabel('Default')
plt.ylabel('Frequency of Purchase')
```

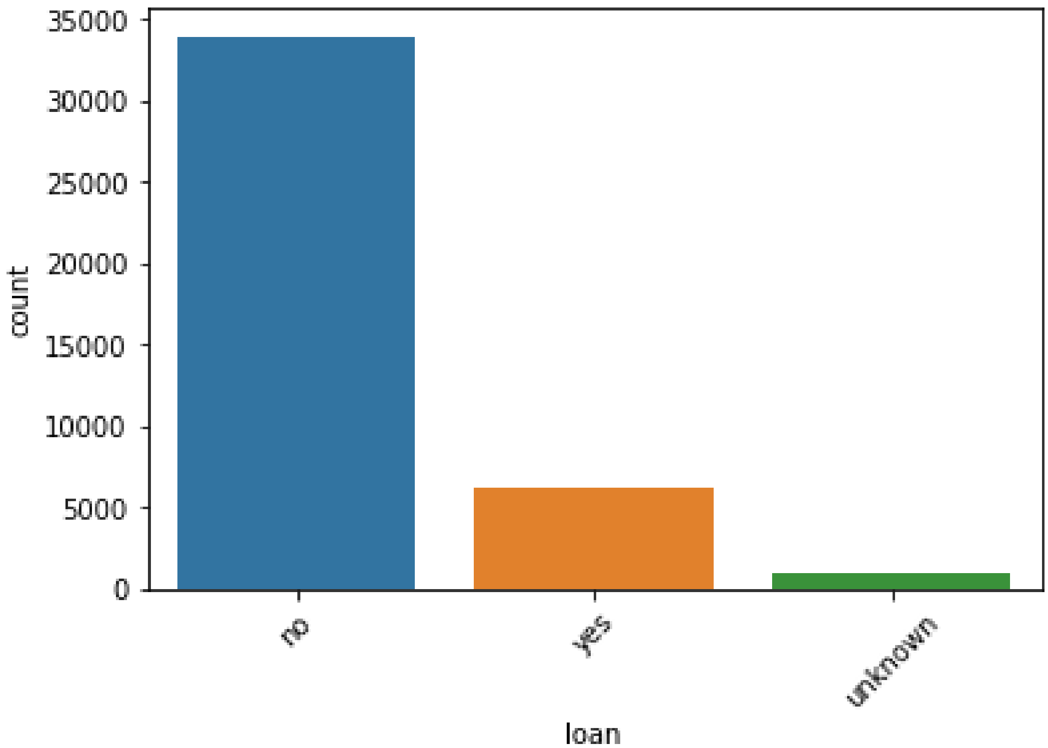
Out[168]:
```
Text(0, 0.5, 'Frequency of Purchase')
```



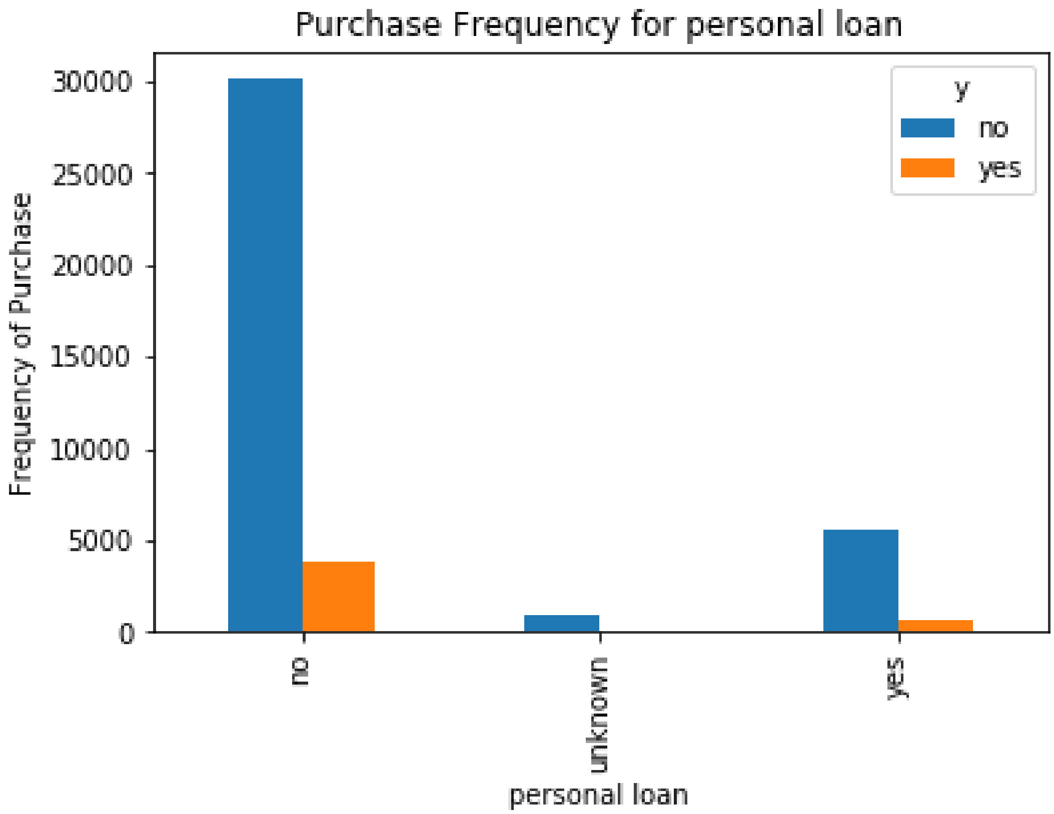
- From this we can infer that Default is an important variable that impacts the target

0.14 Loan

```
In [169]: sns.countplot(x='loan', data= data)
          plt.xticks(rotation=45)

Out[169]: (array([0, 1, 2]), <a list of 3 Text xticklabel objects>)
```



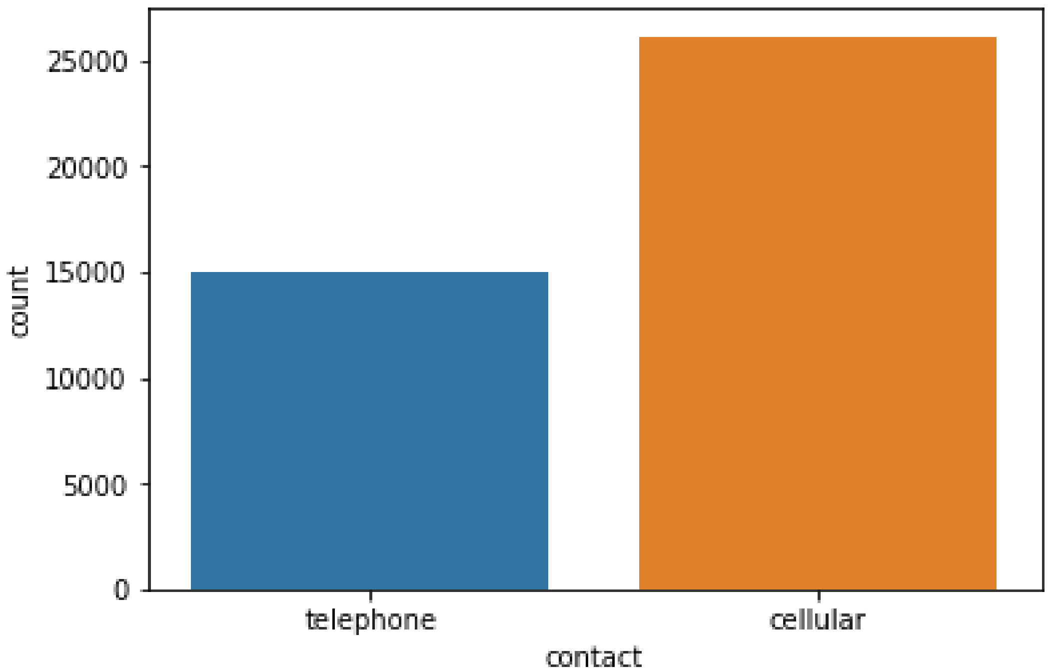

```
In [170]: pd.crosstab(data.loan, data.y).plot(kind='bar')
          plt.title('Purchase Frequency for personal loan')
          plt.xlabel('personal loan')
          plt.ylabel('Frequency of Purchase')

Out[170]: Text(0, 0.5, 'Frequency of Purchase')
```



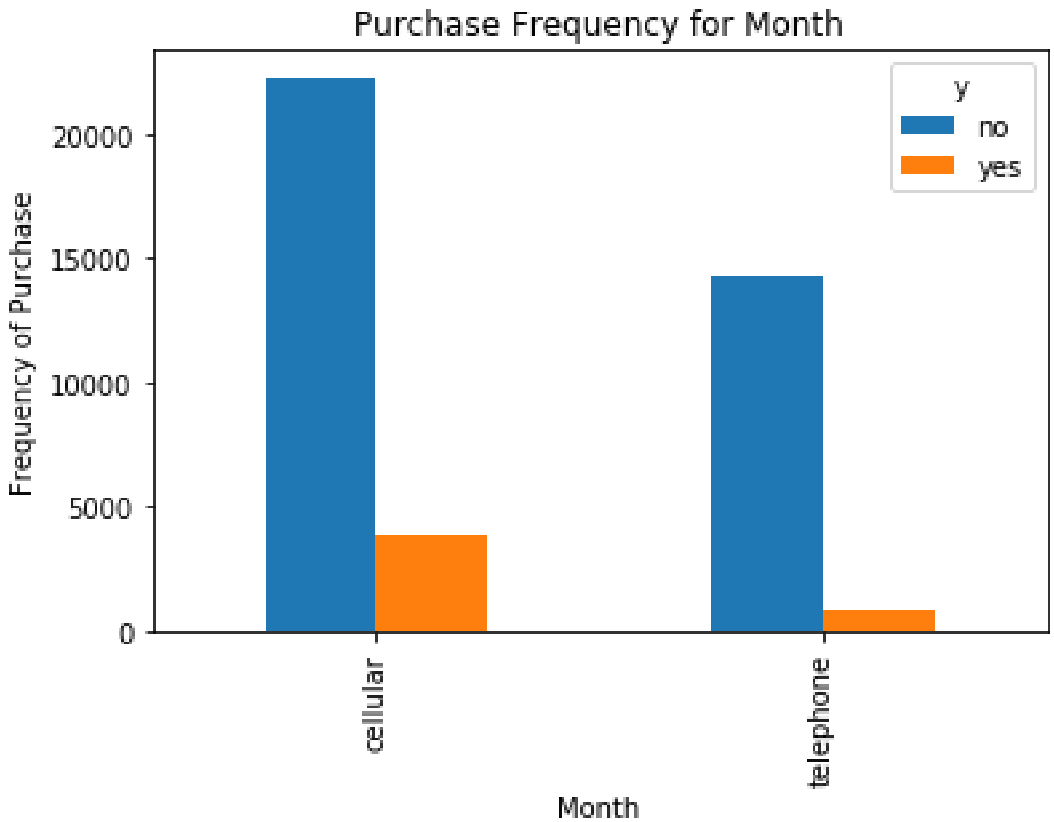
- From this we can infer that having a personal loan impacts the target hence an important variable

0.15 Contact

```
In [171]: sns.countplot(x='contact', data= data)

Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x7f825b23f3c8>
```



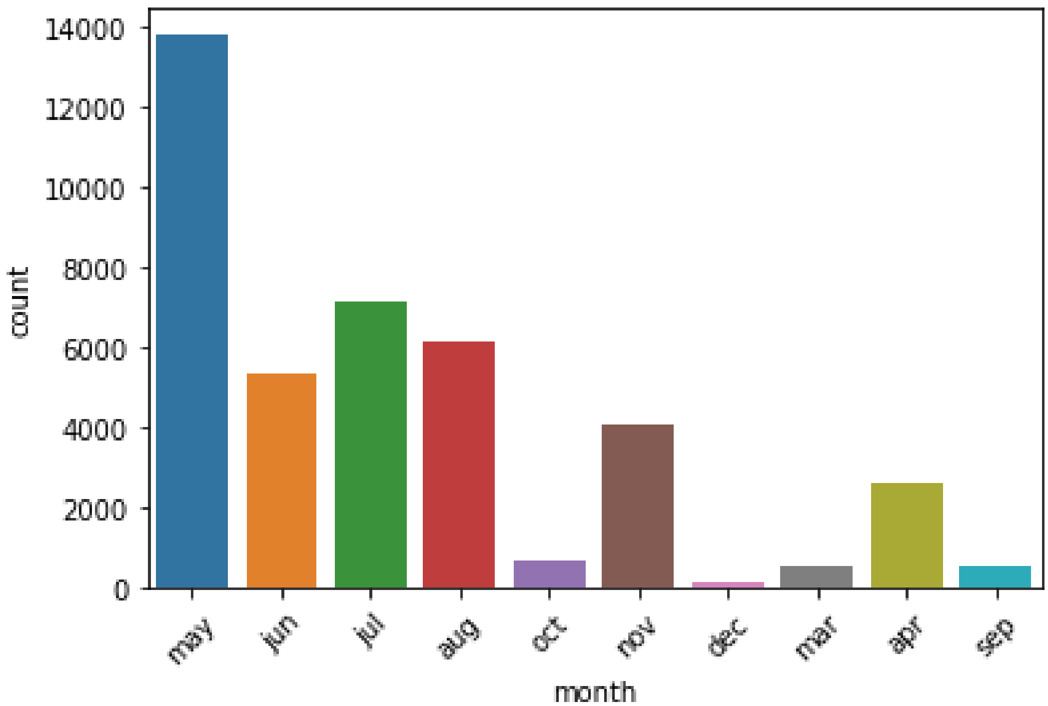

```
In [172]: pd.crosstab(data.contact,data.y).plot(kind='bar')
          plt.title('Purchase Frequency for Month')
          plt.xlabel('Month')
          plt.ylabel('Frequency of Purchase')

Out[172]: Text(0, 0.5, 'Frequency of Purchase')
```



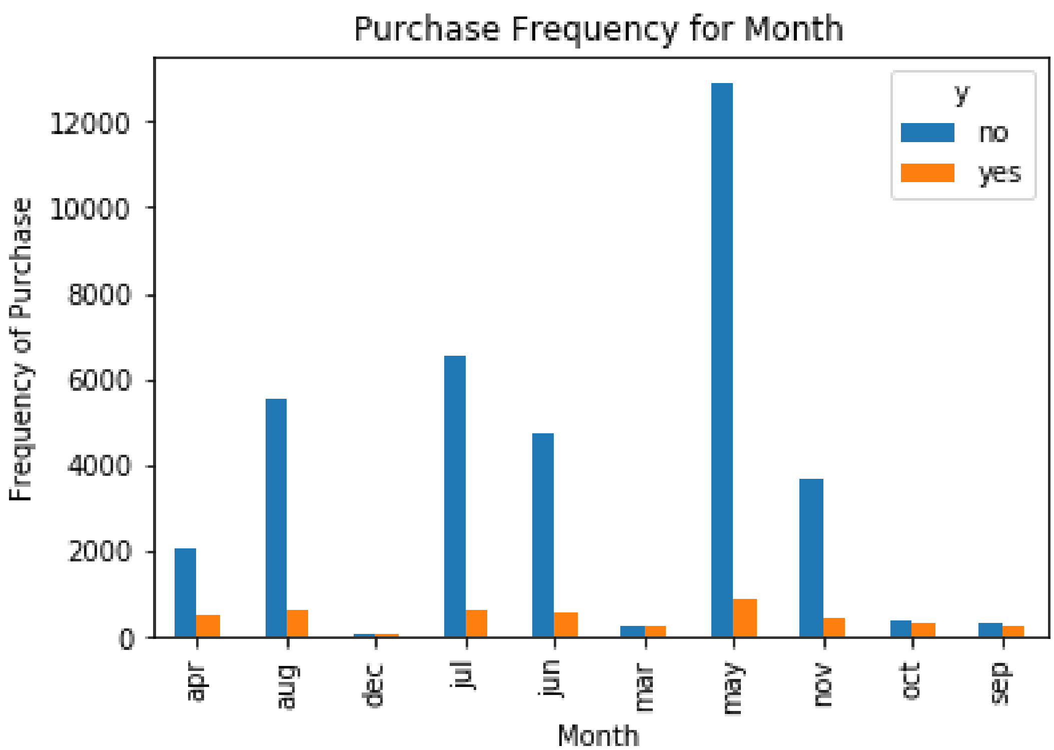
0.16 Month

```
In [173]: sns.countplot(x='month', data= data)
          plt.xticks(rotation=45)
```

```
Out[173]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```




```
In [174]: pd.crosstab(data.month,data.y).plot(kind='bar')
          plt.title('Purchase Frequency for Month')
          plt.xlabel('Month')
          plt.ylabel('Frequency of Purchase')

Out[174]: Text(0, 0.5, 'Frequency of Purchase')
```

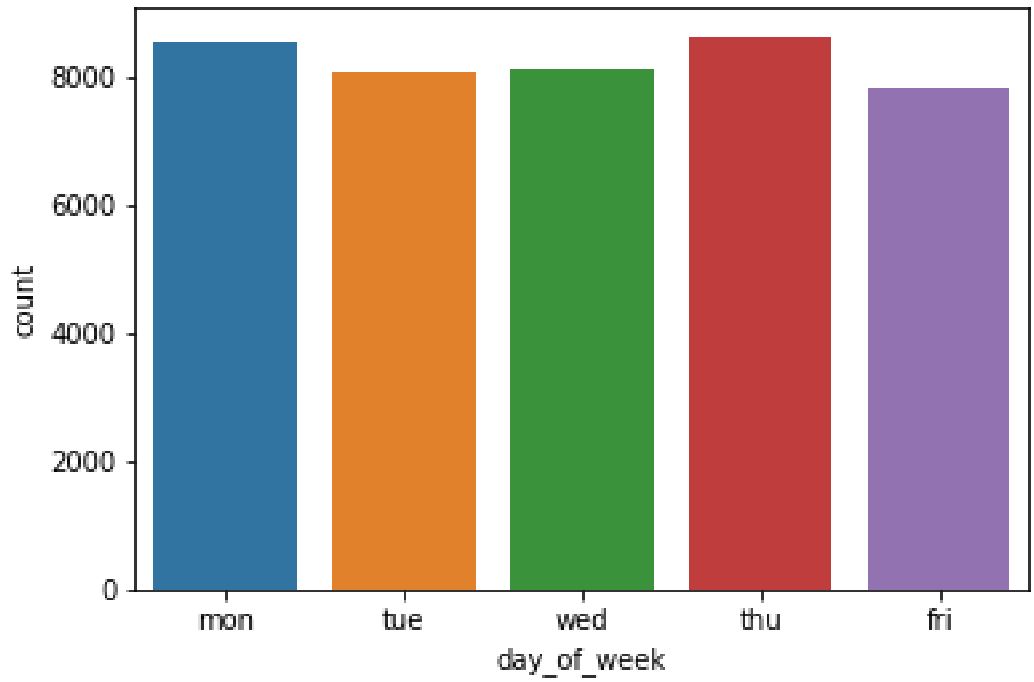


- From this we can infer that month is a good variable that impacts the target

0.17 Day of the Week

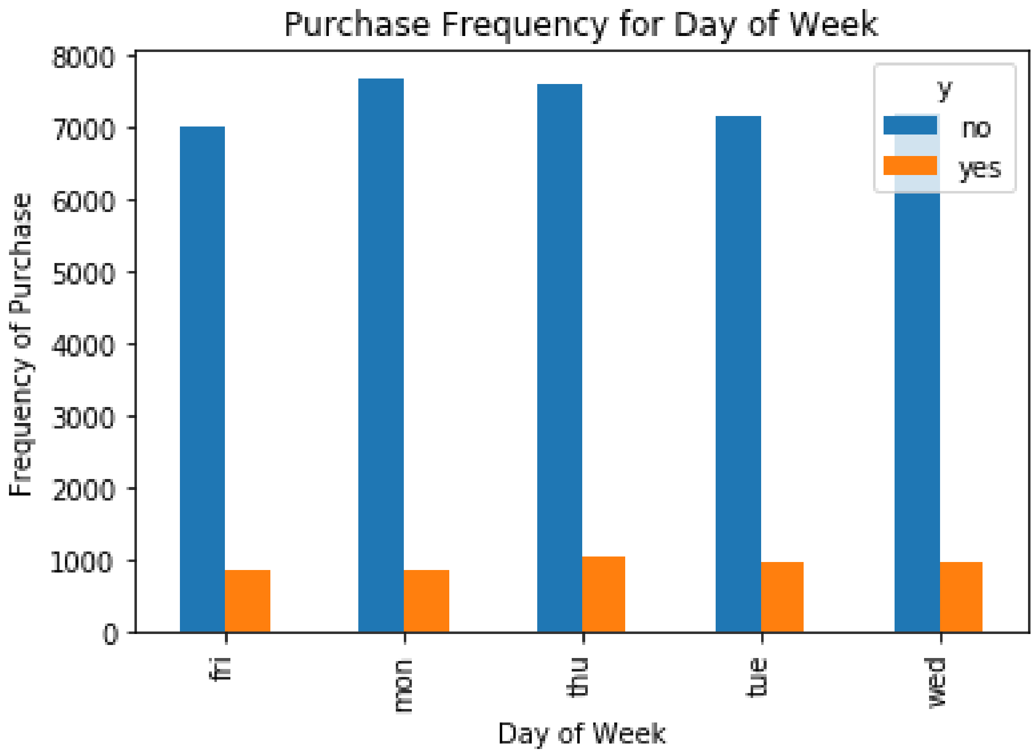
```
In [175]: sns.countplot(x='day_of_week', data= data)
```

```
Out[175]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8259d19be0>
```



In [176]:
```
pd.crosstab(data.day_of_week,data.y).plot(kind='bar')
plt.title('Purchase Frequency for Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Frequency of Purchase')
```

Out[176]:
```
Text(0, 0.5, 'Frequency of Purchase')
```



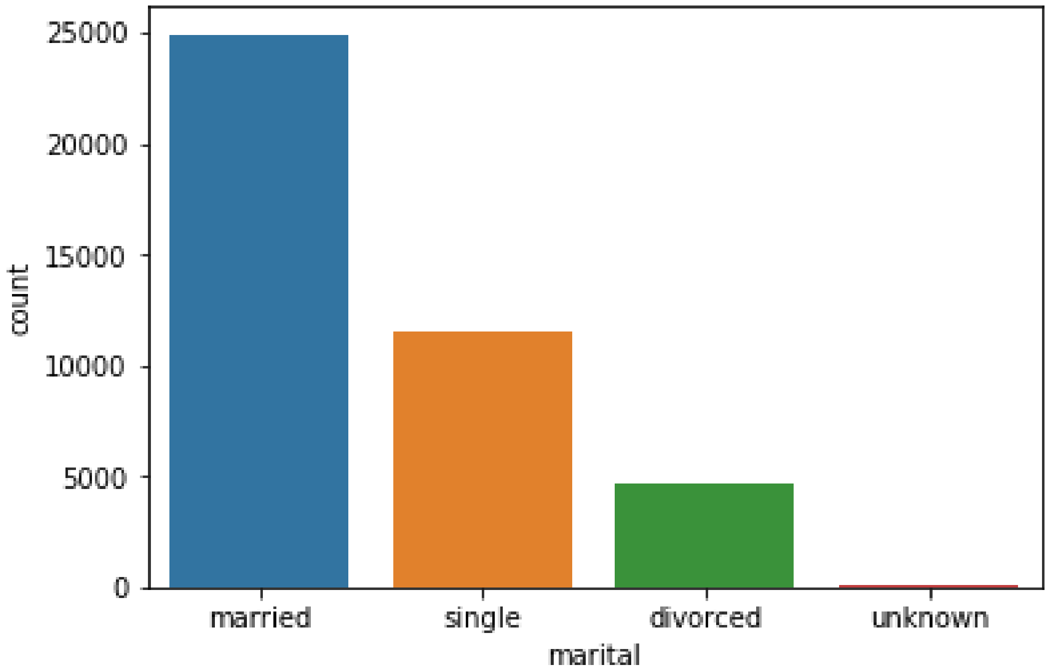
- From this we can infer that day of the week does not impact the target that well hence we can safely remove it

```
In [177]: data.drop(labels='day_of_week', inplace=True, axis=1)
```

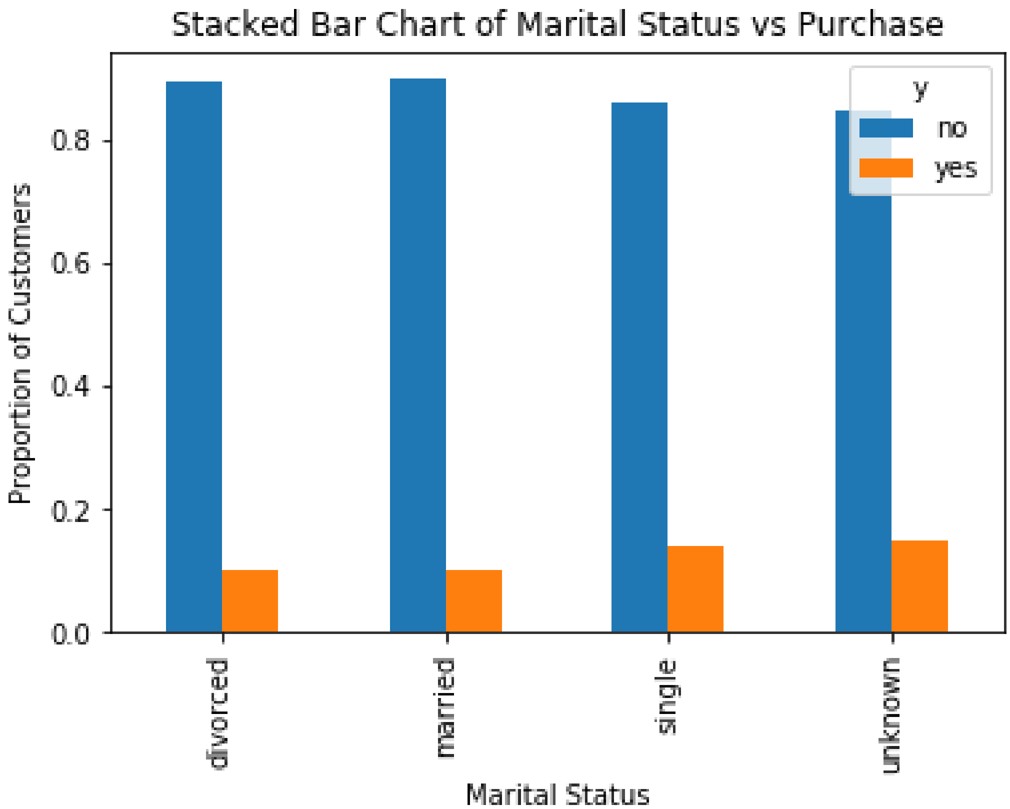
0.18 Marital Status

```
In [178]: sns.countplot(x='marital', data= data)
```

```
Out[178]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8259aa0c88>
```




```
In [179]: table=pd.crosstab(data.marital, data.y)
          table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=False)
          plt.title('Stacked Bar Chart of Marital Status vs Purchase')
          plt.xlabel('Marital Status')
          plt.ylabel('Proportion of Customers')

Out[179]: Text(0, 0.5, 'Proportion of Customers')
```



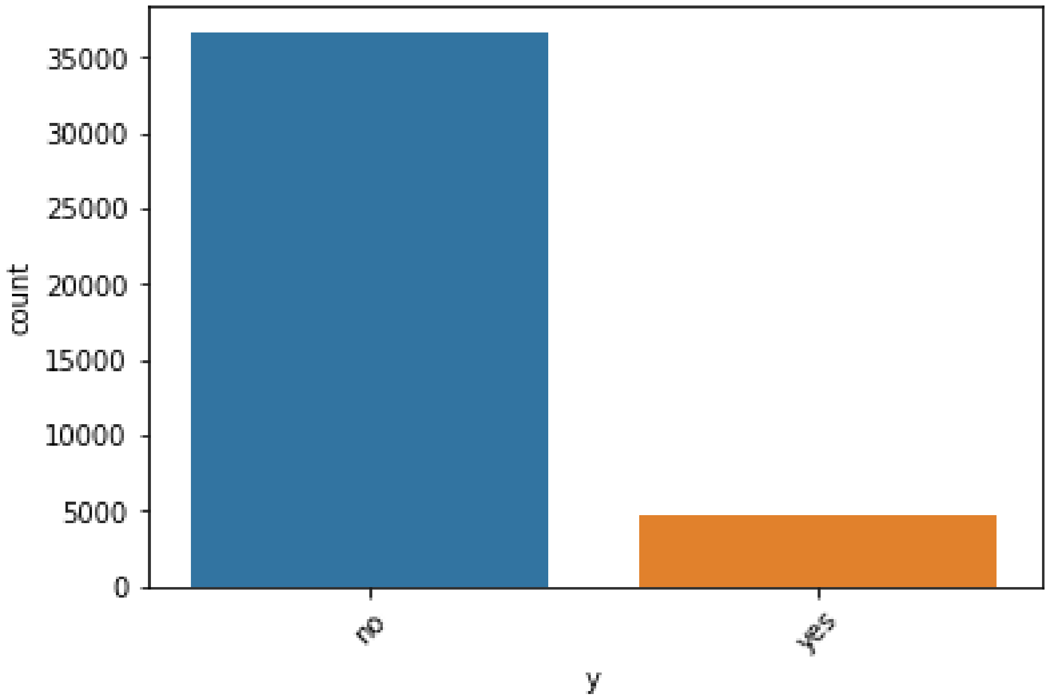
- From this we can infer that marital status does not impact the target that well hence we can remove it (it actually improves the auc by 0.0012 with the advantage of having lesser features)

```
In [180]: data.drop(labels='marital', inplace=True, axis=1)
```

0.19 Countplot for the target variable

- Shows a huge class imbalance which has to be tackled
- Also shows that we value high recall over high precision for the positive class i.e. we do not want to miss customers who have even slight chances of subscribing to the scheme

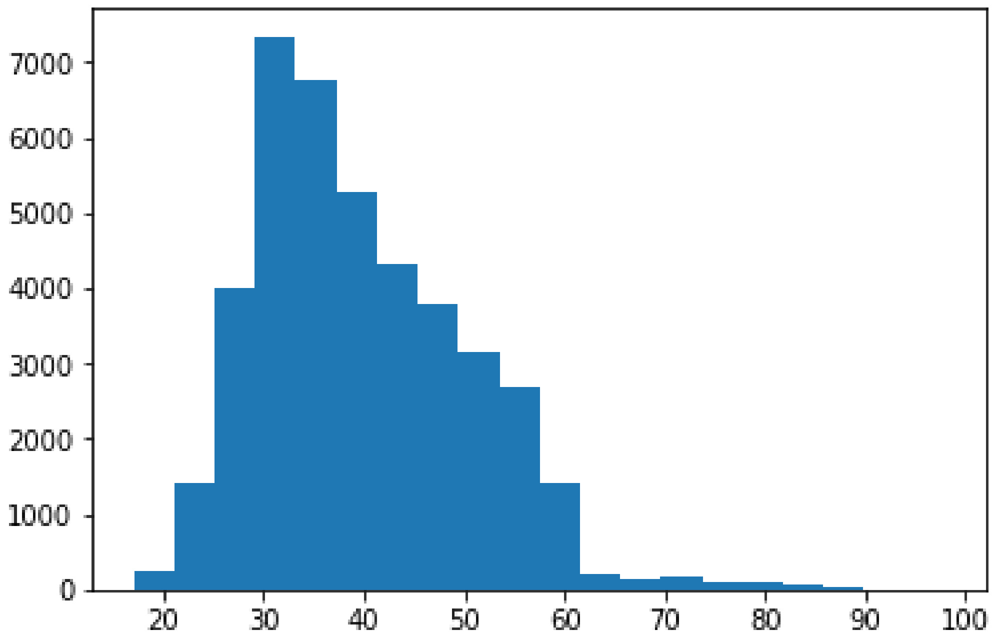
```
In [181]: sns.countplot(x='y', data= data)
          plt.xticks(rotation=45)

Out[181]: (array([0, 1]), <a list of 2 Text xticklabel objects>)
```



0.20 Histograms for numeric features

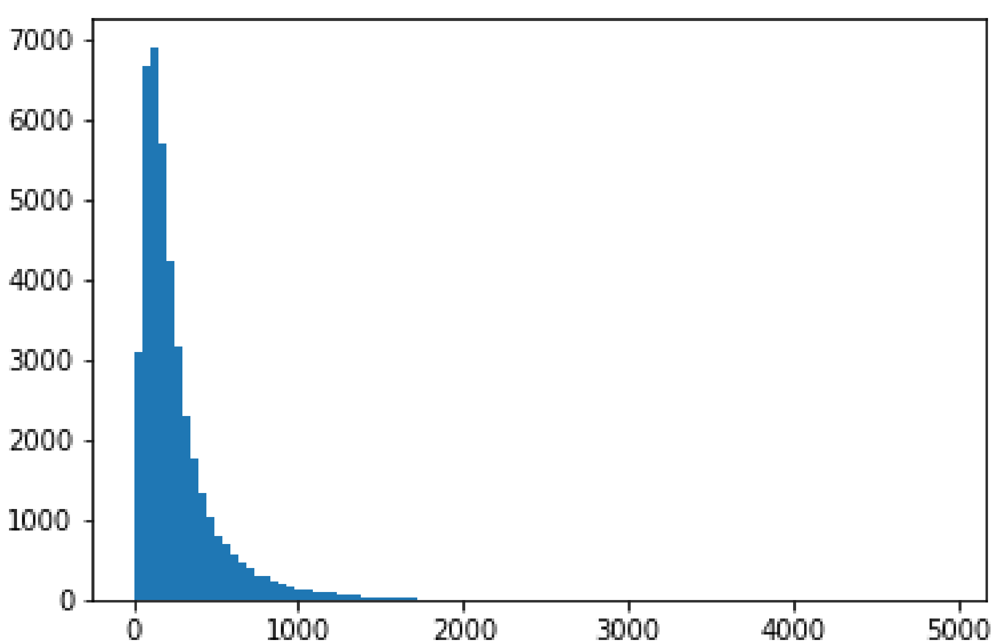
0.21 Age

```
In [182]: plt.hist(data['age'], bins=20)
          plt.show()
```



0.22 Duration

```
In [183]: plt.hist(data['duration'], bins=100)
          plt.show()
```



0.23 Conclusions:

- Average age of people who bought the scheme is higher
- pdays(days since cust was last contacted) is lesser for the customeres who bought
- Campaign(no of client contacts during current campaign) is higher for people who did not buy

In [184]:
```
data.groupby('y').mean()
```

Out[184]:
```
           age    duration  campaign       pdays  previous  emp.var.rate  \
y
no   39.911185  220.844807  2.633085  984.113878  0.132374      0.248875
yes  40.913147  553.191164  2.051724  792.035560  0.492672     -1.233448

     cons.price.idx  cons.conf.idx  euribor3m  nr.employed
y
no        93.603757     -40.593097   3.811491  5176.166600
yes       93.354386     -39.789784   2.123135  5095.115991
```

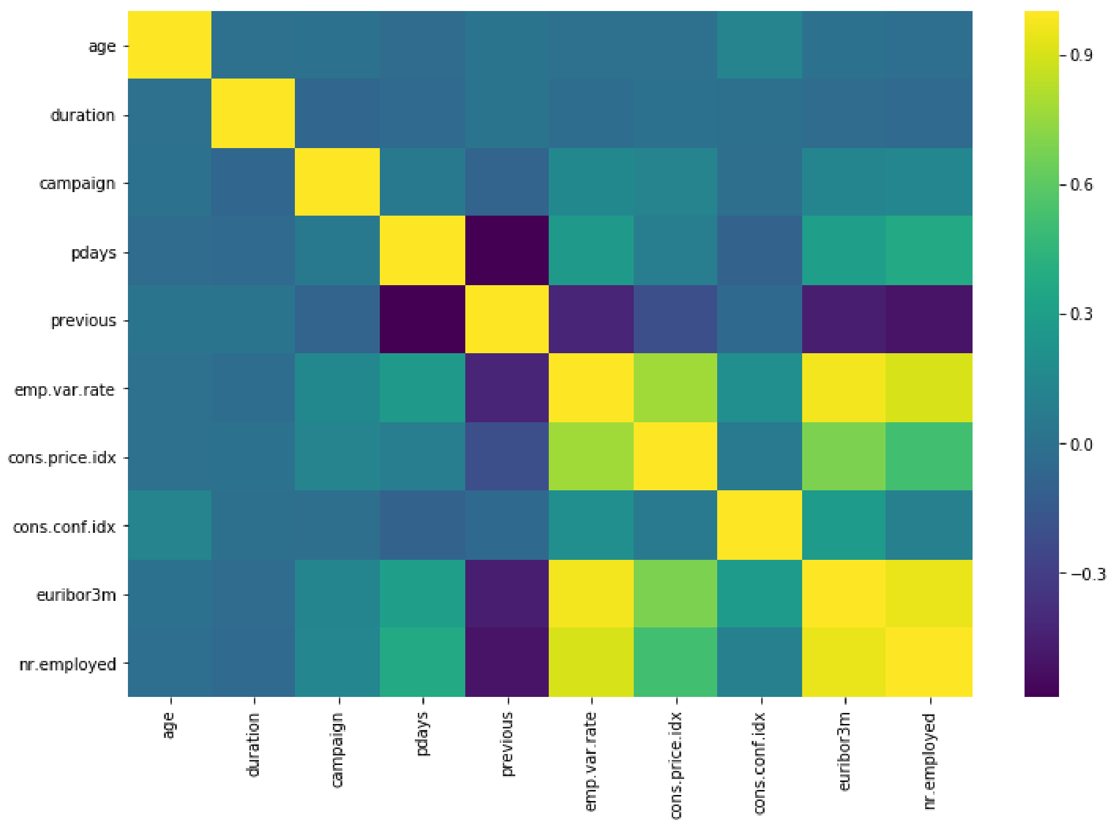
0.24 Multivariate Analysis

0.25 Find the correlation between numeric features

- Remove the highly correlated features
- nr.employed, emp.var.rate and euribor3m are highly correlated with each other, hence we will keep only 1 of them (This is confirmed as we get an increase in auc of about 0.0046)

In [185]:
```
plt.figure(figsize=(12,8))
sns.heatmap(data.corr(), cmap='viridis')
```

Out[185]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8259902080>
```




```
In [186]: data.drop(['emp.var.rate', 'nr.employed'], inplace=True, axis=1)

In [187]: data.columns

Out[187]: Index(['age', 'job', 'education', 'default', 'housing', 'loan', 'contact',
                 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome',
                 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'y'],
                dtype='object')
```

0.26 Implementing a Baseline model

- NO Outlier removal
- K fold cross validation not applied yet
- Converting all categorical variables to One-Hot-Encoding with dummy columns

0.27 Dividing the data into features and target variable

```
In [188]: X= data.iloc[:, :-1]
          y= data.iloc[:, -1]
```

0.28 Create dummy features as One-Hot encodings

In [189]:
```
# Get dummy columns
X = pd.get_dummies(X, prefix_sep='_', drop_first=True)

# Converting target column to numeric representation
y= y.map(dict(yes=1, no=0))
```

In [190]:
```
y= pd.DataFrame(y)
y.tail()
```

Out[190]:

	y
41183	1
41184	0
41185	0
41186	1
41187	0

In [191]:
```
print("Shape of X:", X.shape)
X.head()
```

```
Shape of X: (41188, 44)
```

Out[191]:

	age	duration	campaign	pdays	previous	cons.price.idx	cons.conf.idx	\
0	56	261	1	999	0	93.994	-36.4	
1	57	149	1	999	0	93.994	-36.4	
2	37	226	1	999	0	93.994	-36.4	
3	40	151	1	999	0	93.994	-36.4	
4	56	307	1	999	0	93.994	-36.4	

	euribor3m	job_blue-collar	job_entrepreneur	...	month_dec	month_jul	\
0	4.857	0	0	...	0	0	
1	4.857	0	0	...	0	0	
2	4.857	0	0	...	0	0	
3	4.857	0	0	...	0	0	
4	4.857	0	0	...	0	0	

	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	\
0	0	0	1	0	0	0	
1	0	0	1	0	0	0	
2	0	0	1	0	0	0	
3	0	0	1	0	0	0	
4	0	0	1	0	0	0	

	poutcome_nonexistent	poutcome_success
0	1	0
1	1	0
2	1	0

```
3                      1                  0
4                      1                  0

[5 rows x 44 columns]
```

0.29 Dividing the dataset into training and testing data

```
In [192]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
In [193]: columns = X_train.columns
```

```
In [194]: columns
```

```
Out[194]: Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'cons.price.idx',
                 'cons.conf.idx', 'euribor3m', 'job_blue-collar', 'job_entrepreneur',
                 'job_housemaid', 'job_management', 'job_retired', 'job_self-employed',
                 'job_services', 'job_student', 'job_technician', 'job_unemployed',
                 'job_unknown', 'education_basic.6y', 'education_basic.9y',
                 'education_high.school', 'education_illiterate',
                 'education_professional.course', 'education_university.degree',
                 'education_unknown', 'default_unknown', 'default_yes',
                 'housing_unknown', 'housing_yes', 'loan_unknown', 'loan_yes',
                 'contact_telephone', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
                 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
                 'poutcome_nonexistent', 'poutcome_success'],
                dtype='object')
```

1 Handling Class Imbalance

1.1 Using SMOTE to perform oversampling on the minority class

```
In [195]: print('Before OverSampling, the shape of X_train: {}'.format(X_train.shape))
          print('Before OverSampling, the shape of y_train: {} \n'.format(y_train.shape))

          print("Before OverSampling, counts of label '1': {0}".format((y_train == 1).sum()))
          print("Before OverSampling, counts of label '0': {} \n".format((y_train == 0).sum())

          sm = SMOTE(random_state=2)
          X_train_sm, y_train_sm = sm.fit_sample(X_train, y_train)

          print('After OverSampling, the shape of X_train: {}'.format(X_train_sm.shape))
          print('After OverSampling, the shape of y_train: {} \n'.format(y_train_sm.shape))

          print("Training Data")
          print("After OverSampling, counts of label '1': {}".format(sum(y_train_sm == 1)))
          print("After OverSampling, counts of label '0': {}".format(sum(y_train_sm == 0)))

          X_train_os = pd.DataFrame(data= X_train_sm, columns= columns)
```

```
y_train_os = pd.DataFrame(data= y_train_sm, columns=['y'])
X_train_os.head()
```

```
Before OverSampling, the shape of X_train: (28831, 44)
Before OverSampling, the shape of y_train: (28831, 1)

Before OverSampling, counts of label '1': y    3223
dtype: int64
Before OverSampling, counts of label '0': y    25608
dtype: int64



/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)


After OverSampling, the shape of X_train: (51216, 44)
After OverSampling, the shape of y_train: (51216,)

Training Data
After OverSampling, counts of label '1': 25608
After OverSampling, counts of label '0': 25608
```

Out[195]:

```
    age  duration  campaign  pdays  previous  cons.price.idx  cons.conf.idx  \
0  38.0     431.0       4.0  999.0       0.0          93.444          -36.1
1  51.0     115.0       3.0  999.0       0.0          94.215          -40.3
2  29.0     223.0       5.0  999.0       0.0          93.994          -36.4
3  40.0     240.0       2.0  999.0       0.0          93.918          -42.7
4  33.0      16.0      18.0  999.0       0.0          94.465          -41.8

   euribor3m  job_blue-collar  job_entrepreneur  ...  month_dec  month_jul  \
0      4.963              0.0               0.0  ...        0.0        0.0
1      0.896              0.0               0.0  ...        0.0        1.0
2      4.857              1.0               0.0  ...        0.0        0.0
3      4.962              1.0               0.0  ...        0.0        1.0
4      4.959              1.0               0.0  ...        0.0        0.0

   month_jun  month_mar  month_may  month_nov  month_oct  month_sep  \
0        0.0        0.0        0.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0
2        0.0        0.0        1.0        0.0        0.0        0.0
3        0.0        0.0        0.0        0.0        0.0        0.0
4        1.0        0.0        0.0        0.0        0.0        0.0

   poutcome_nonexistent  poutcome_success
0                   1.0               0.0
1                   1.0               0.0
```

```
2                    1.0              0.0
3                    1.0              0.0
4                    1.0              0.0

[5 rows x 44 columns]
```

1.2 class imbalance for test data

```
In [196]: print("counts of label '1': {0}".format((y_test == 1).sum()))
          print("counts of label '0': {} \n".format((y_test == 0).sum()))
```

```
counts of label '1': y    1417
dtype: int64
counts of label '0': y    10940
dtype: int64
```

1.3 Using Logistic regression model as a baseline

```
In [197]: logreg = LogisticRegression()
          logreg.fit(X_train_os, y_train_os)
```

```
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433:
  FutureWarning)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data
  y = column_or_1d(y, warn=True)
```

```
Out[197]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

1.4 Testing the trained model

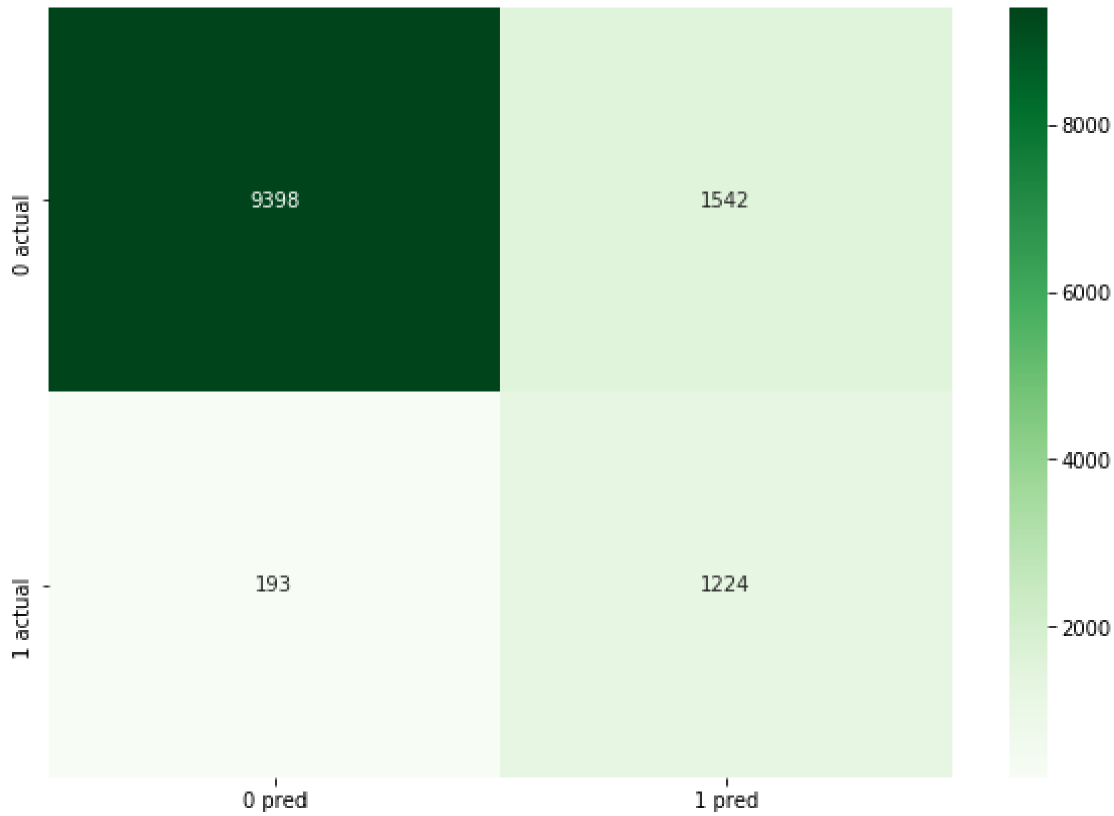
- Accuracy is a bad metric since we have a class imbalance in the data
- Use ROC_AUC instead

```
In [198]: y_pred = logreg.predict_proba(X_test)[:, 1]
          print("Predicted probabilities for the positive class:", y_pred)

          auc = roc_auc_score(y_test, y_pred)
          print('AUC: %.4f' % auc)

          # print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logr
          y_pred= np.where(y_pred>0.5, 1, 0)
          print(y_pred)
```

```
Predicted probabilities for the positive class: [0.01417583 0.06137143 0.73633301 ... 0.022144
AUC: 0.9298
[0 0 1 ... 0 0 1]
```

In [199]:
```python
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm)
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g', cmap= 'Greens', xticklabels=['0 pred', '1 pre
```

Out[199]: <matplotlib.axes._subplots.AxesSubplot at 0x7f82598eb828>



- In this confusion matrix we have to minimise the '98' as much as possible and maximise the '821'

In [200]:
```python
df_cm
```

Out[200]:

	0	1
0	9398	1542
1	193	1224

1.5 Classification Report

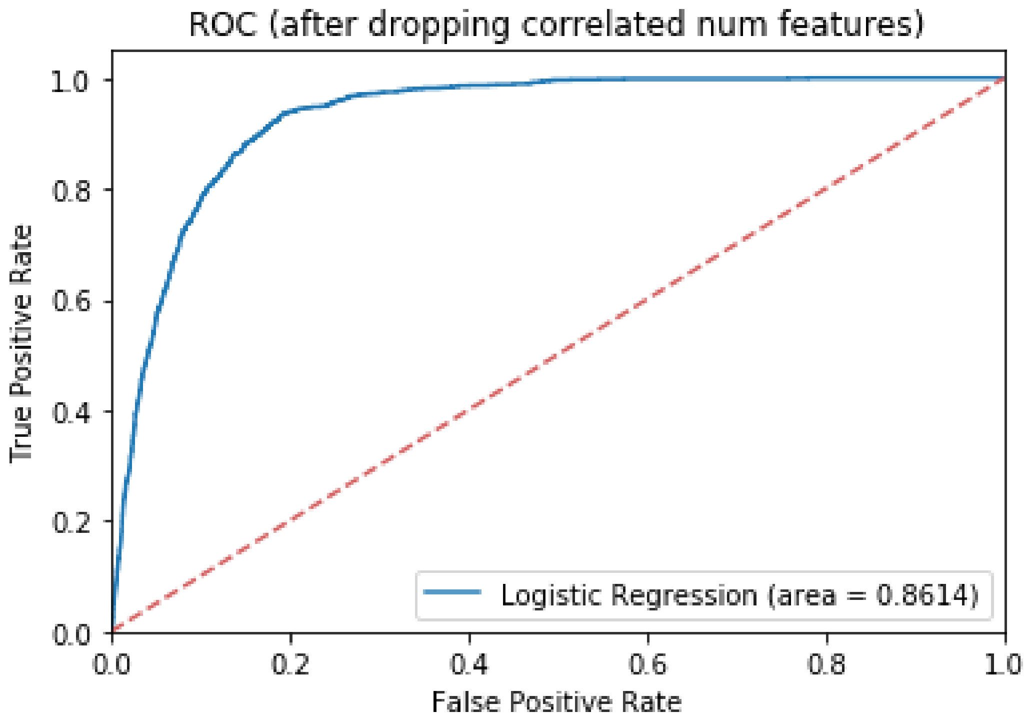
```
In [202]: print(classification_report(y_test, y_pred, target_names=['no', 'yes']))
```

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.92     10940
           1       0.44      0.86      0.59      1417

   micro avg       0.86      0.86      0.86     12357
   macro avg       0.71      0.86      0.75     12357
weighted avg       0.92      0.86      0.88     12357
```

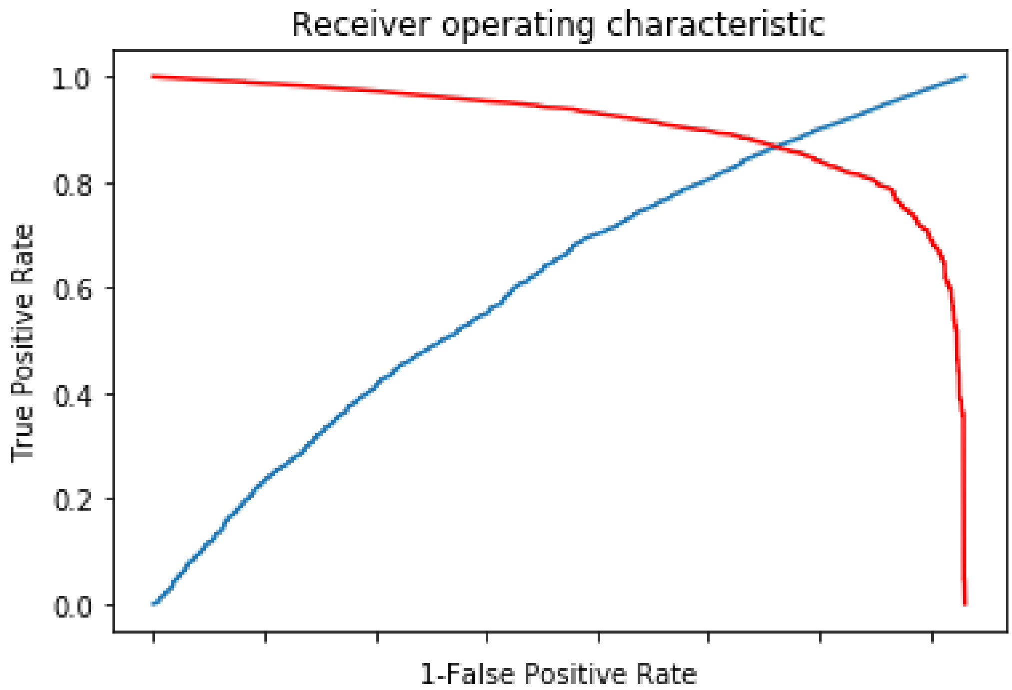
```
In [203]: recall= m.recall_score(y_test, y_pred)
          print('Recall: %.4f' %recall)
```

```
Recall: 0.8638
```

```
In [204]: logit_roc_auc = roc_auc_score(y_test, y_pred)
          fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
          plt.figure()
          plt.plot(fpr, tpr, label='Logistic Regression (area = %0.4f)' % logit_roc_auc)
          plt.plot([0, 1], [0, 1],'r--')
          plt.xlim([0.0, 1.0])
          plt.ylim([0.0, 1.05])
          plt.xlabel('False Positive Rate')
          plt.ylabel('True Positive Rate')
          plt.title('ROC (after dropping correlated num features)')
          plt.legend(loc="lower right")
          plt.savefig('Log_ROC')
          plt.show()
```




```
In [151]: i = np.arange(len(tpr)) # index for df
          roc = pd.DataFrame({'fpr' : pd.Series(fpr, index=i),'tpr' : pd.Series(tpr, index = i)
          roc.iloc[(roc.tf-0).abs().argsort()[:1]]

          # Plot tpr vs 1-fpr
          fig, ax = plt.subplots()
          plt.plot(roc['tpr'])
          plt.plot(roc['1-fpr'], color = 'red')
          plt.xlabel('1-False Positive Rate')
          plt.ylabel('True Positive Rate')
          plt.title('Receiver operating characteristic')
          ax.set_xticklabels([])

Out[151]: []
```



In [205]:
```
pred= logreg.predict_proba(X_test)
pred_df= pd.DataFrame(pred).iloc[:, 1]
fpr, tpr, thresholds = m.roc_curve(y_test, pred_df)
m.auc(fpr, tpr)
```

Out[205]: 0.9298495418004669

1.6 Different classifiers

In [64]:
```
classifiers = {'Gradient Boosting Classifier':GradientBoostingClassifier(),
               'Adaptive Boosting Classifier':AdaBoostClassifier(),
               'Linear Discriminant Analysis':LinearDiscriminantAnalysis(),'Logistic I
               'Random Forest Classifier': RandomForestClassifier(),
               'K Nearest Neighbour':KNeighborsClassifier(8),
               'Gaussian Naive Bayes Classifier':GaussianNB(),
                }

log_cols = ["Classifier", "Accuracy","Precision Score","Recall Score","F1-Score","roc-
log = pd.DataFrame(columns=log_cols)
```

In [65]:
```
for name,classify in classifiers.items():

    cls = classify
    cls =cls.fit(X_train_os, y_train_os)
```

```
            y_out = cls.predict(X_test)

            accuracy = m.accuracy_score(y_test, y_out)
            precision = m.precision_score(y_test, y_out, average='macro')
            recall = m.recall_score(y_test, y_out, average='macro')
            roc_auc = roc_auc_score(y_test, y_out)
            f1_score = m.f1_score(y_test, y_out, average='macro')

            log_entry = pd.DataFrame([[name, accuracy, precision, recall, f1_score, roc_auc]]
            log = log.append(log_entry)
```

```
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388:
  warnings.warn("Variables are collinear.")
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433:
  FutureWarning)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: Future
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/ipykernel_launcher.py:4: DataConversio
  after removing the cwd from sys.path.
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/ipykernel_launcher.py:4: DataConversio
  after removing the cwd from sys.path.
/home/nikhil/.virtualenvs/DL/lib/python3.6/site-packages/sklearn/utils/validation.py:761: Data(
  y = column_or_1d(y, warn=True)
```

1.7 Hence we see that Logistic Regression Classifier gives the best result: auc_roc= 0.8767

In [66]:
```
print(log)
plt.xlabel('Accuracy')
plt.title('Classifier Accuracy')
sns.set_color_codes("muted")
sns.barplot(x='roc-auc_Score', y='Classifier', data=log, color="r")
plt.show()
```

```
                      Classifier  Accuracy  Precision Score  Recall Score  \
0     Gradient Boosting Classifier  0.907583         0.765472      0.770877
0     Adaptive Boosting Classifier  0.906288         0.763798      0.743337
0     Linear Discriminant Analysis  0.862345         0.705589      0.844702
0              Logistic Regression  0.862102         0.708696      0.859248
```

0	Random Forest Classifier	0.905721	0.770322	0.690353
0	K Nearest Neighbour	0.847940	0.682869	0.804682
0	Gaussian Naive Bayes Classifier	0.845917	0.653413	0.712258

	F1-Score	roc-auc_Score
0	0.768135	0.770877
0	0.752968	0.743337
0	0.743986	0.844702
0	0.748399	0.859248
0	0.720642	0.690353
0	0.715914	0.804682
0	0.674206	0.712258