Assg1_1

April 28, 2019

```
In [77]: %matplotlib inline

         import matplotlib.pyplot as plt
         import seaborn as sns
         import pandas as pd
         import numpy as np
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV

         from sklearn.metrics import accuracy_score
         from sklearn import metrics as m
         from sklearn.model_selection import StratifiedShuffleSplit
         from sklearn.metrics import roc_auc_score

         from sklearn.svm import LinearSVC
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.neural_network import MLPClassifier

         from sklearn.metrics import roc_curve
         from sklearn.metrics import auc
         from sklearn.metrics import classification_report,confusion_matrix

         import tqdm as tqdm
```

```
In [78]: import warnings
         warnings.filterwarnings("ignore")
```

Helper functions

```
In [79]: # decode categories from encoded values
         def getCategories(attribute):
             print("{} categories".format(attribute))
             categories = df[attribute].astype('category').cat.categories
             for x, y in zip(categories, [l for l in range( 1, len(categories) + 1)] ):
                 print(y, x)

         def roc(classifiers, X_train, y_train, X_test, y_test):
```

```
    for name,classifier in classifiers.items():
        cls = classifier
        cls =cls.fit(X_train,y_train)

        y_pred = cls.predict(X_test)

        y_true=sorted(y_test)
        y_score=sorted(y_pred)
        # Compute fpr, tpr, thresholds and roc auc
        fpr, tpr, thresholds = roc_curve(y_true, y_score)
        roc_auc = auc(fpr, tpr)

        # Plot ROC curve
        plt.plot(fpr, tpr, label='ROC curve (area = %0.3f)' % roc_auc)
        plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.xlabel('False Positive Rate or (1 - Specifity)')
        plt.ylabel('True Positive Rate or (Sensitivity)')
        plt.title('Receiver Operating Characteristic')
        plt.legend(loc="lower right")

def classify_and_compare(classifiers, X_train, y_train, X_test, y_test):
    metric_columns = ["Classifier", "Accuracy","Precision Score","Recall Score","F1-S
    df_metric = pd.DataFrame(columns=metric_columns)

    for name,classifier in classifiers.items():
        cls = classifier
        cls =cls.fit(X_train,y_train)

        y_pred = cls.predict(X_test)

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

        metric_entry = pd.DataFrame([[name,accuracy,precision,recall,f1_score,roc_auc
        df_metric = df_metric.append(metric_entry)

        print(name)
        print(classification_report(y_test, y_pred))


    print(df_metric)
    plt.xlabel('Accuracy')
    plt.title('Classifier Accuracy')
```

```
    sns.set_color_codes("muted")
    sns.barplot(x='Accuracy', y='Classifier', data=df_metric, color="g")
    plt.show()

def findBestParamters(clf, parameters, X_train, y_train):
    clf =GridSearchCV(clf, parameters)
    clf.fit(X_train, y_train)

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

```
In [80]: df = pd.read_csv('./dataset/[UCI]bank-additional/bank-additional/bank-additional-full

         df_x = df.copy()
         del df_x['y']

         df_y = df[['y']].copy().astype('category')
         # label encode y - yes->1 no->0
         df_y = df_y.apply(lambda x: x.cat.codes)

         df.head()
```

```
Out[80]:    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]
```

---

1 Analysis

```
In [81]: # shape of the feature dataframe
         print(df_x.shape)
```

```
(41188, 20)
```

```
In [82]: # features
         print(list(df_x.columns))
```

```
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_c
```

```
In [83]: print(df_x.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 20 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
dtypes: float64(5), int64(5), object(10)
memory usage: 6.3+ MB
None
```

- All the features are having 41188 non-null entries and hence there are no null values
- There are 10 continuos and 10 categorical attributes

Make the categorical variables of dtype 'category' and label encode

```
In [84]: for col in df_x.select_dtypes(include=['object']):
             # Changing the dtype of categorical columns to 'category' reduces memory usage
             df_x[col] = df_x[col].astype('category')

         categorical_attrs = list(df_x.select_dtypes(include=['category']).copy().columns)

         df_x[categorical_attrs] = df_x[categorical_attrs].apply(lambda x: x.cat.codes)

         print(getCategories('education'))
         df_x.head(2)
```

```
education categories
1 basic.4y
2 basic.6y
3 basic.9y
4 high.school
5 illiterate
6 professional.course
7 university.degree
8 unknown
None
```

```
Out[84]:    age  job  marital  education  default  housing  loan  contact  month  \
         0   56    3        1          0        0        0     0        1      6
         1   57    7        1          3        1        0     0        1      6

            day_of_week  duration  campaign  pdays  previous  poutcome  emp.var.rate  \
         0            1       261         1    999         0         1           1.1
         1            1       149         1    999         0         1           1.1

            cons.price.idx  cons.conf.idx  euribor3m  nr.employed
         0          93.994          -36.4      4.857       5191.0
         1          93.994          -36.4      4.857       5191.0
```

```
In [85]: print(df_x.info(verbose=False))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Columns: 20 entries, age to nr.employed
dtypes: float64(5), int64(5), int8(10)
memory usage: 3.5 MB
None
```

Memory usage decreased from 6.3+MB to 3.5MB by making the categorical attributes datatype as **category**

```
In [86]: labels = ["Opened term subscription", "Didn't open term subscription"]
         df_y['y'].value_counts().plot.pie(labels=labels, explode=[0, 0.1]);
```

There are more records with the output of not opening a term subscription

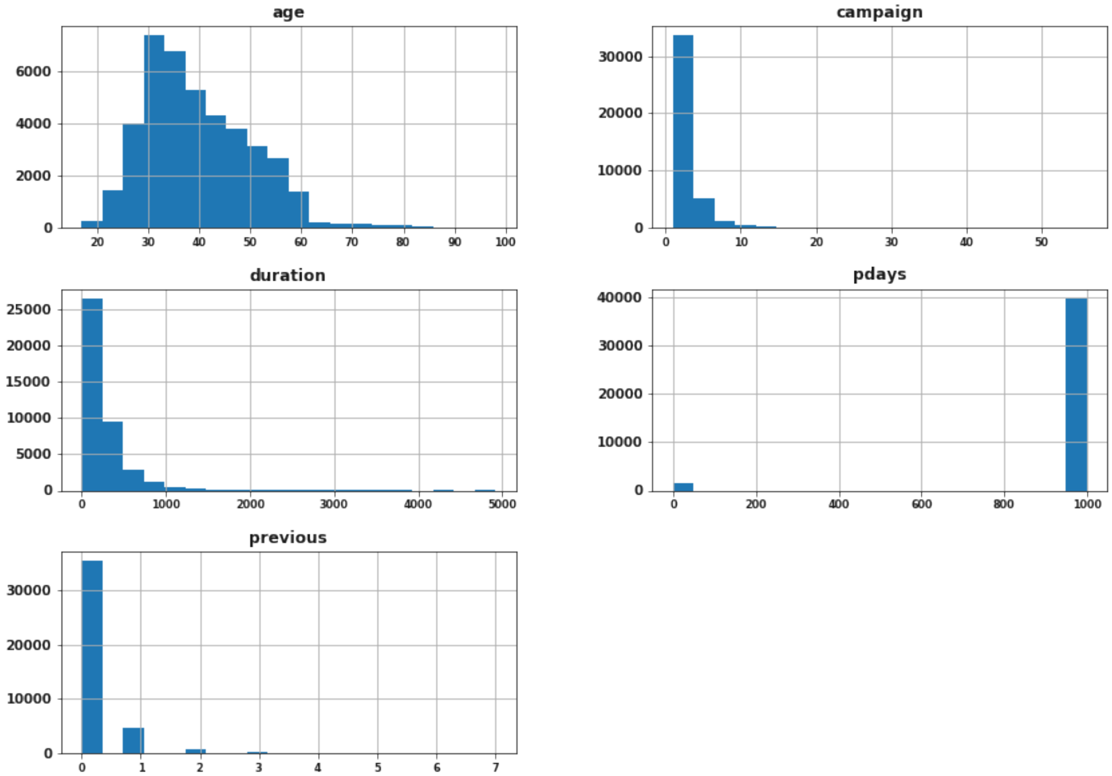
```
In [87]: print(df_x.describe())
```

```
                age          job       marital     education       default  \
count  41188.00000  41188.00000  41188.000000  41188.000000  41188.000000
mean      40.02406      3.72458      1.172769      3.747184      0.208872
std       10.42125      3.59456      0.608902      2.136482      0.406686
min       17.00000      0.00000      0.000000      0.000000      0.000000
25%       32.00000      0.00000      1.000000      2.000000      0.000000
50%       38.00000      2.00000      1.000000      3.000000      0.000000
75%       47.00000      7.00000      2.000000      6.000000      0.000000
max       98.00000     11.00000      3.000000      7.000000      2.000000

            housing          loan       contact         month   day_of_week  \
count  41188.000000  41188.000000  41188.000000  41188.000000  41188.000000
mean       1.071720      0.327425      0.365252      4.230868      2.004613
std        0.985314      0.723616      0.481507      2.320025      1.397575
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      3.000000      1.000000
50%        2.000000      0.000000      0.000000      4.000000      2.000000
75%        2.000000      0.000000      1.000000      6.000000      3.000000
max        2.000000      2.000000      1.000000      9.000000      4.000000

           duration      campaign         pdays      previous      poutcome  \
count  41188.000000  41188.000000  41188.000000  41188.000000  41188.000000
mean     258.285010      2.567593    962.475454      0.172963      0.930101
std      259.279249      2.770014    186.910907      0.494901      0.362886
min        0.000000      1.000000      0.000000      0.000000      0.000000
25%      102.000000      1.000000    999.000000      0.000000      1.000000
```

```
50%       180.000000      2.000000    999.000000      0.000000      1.000000
75%       319.000000      3.000000    999.000000      0.000000      1.000000
max      4918.000000     56.000000    999.000000      7.000000      2.000000

       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
```

- Average age of the people in the dataset is 40 with std of 10.42
- Min. age is 17
- Max. age is 98
- 75% of the people have are aged below 47
- 47 * 1.5 = 70 => ages above 70 are outliers
- Average duration of the people speaking in the dataset is 258s with std of 259s
- Min. duration is 0s
- Max. duration is 4918s
- 75% of the people spoke for less than 319s
- 319 * 1.5 = 478.5 => above 478.5s are outliers

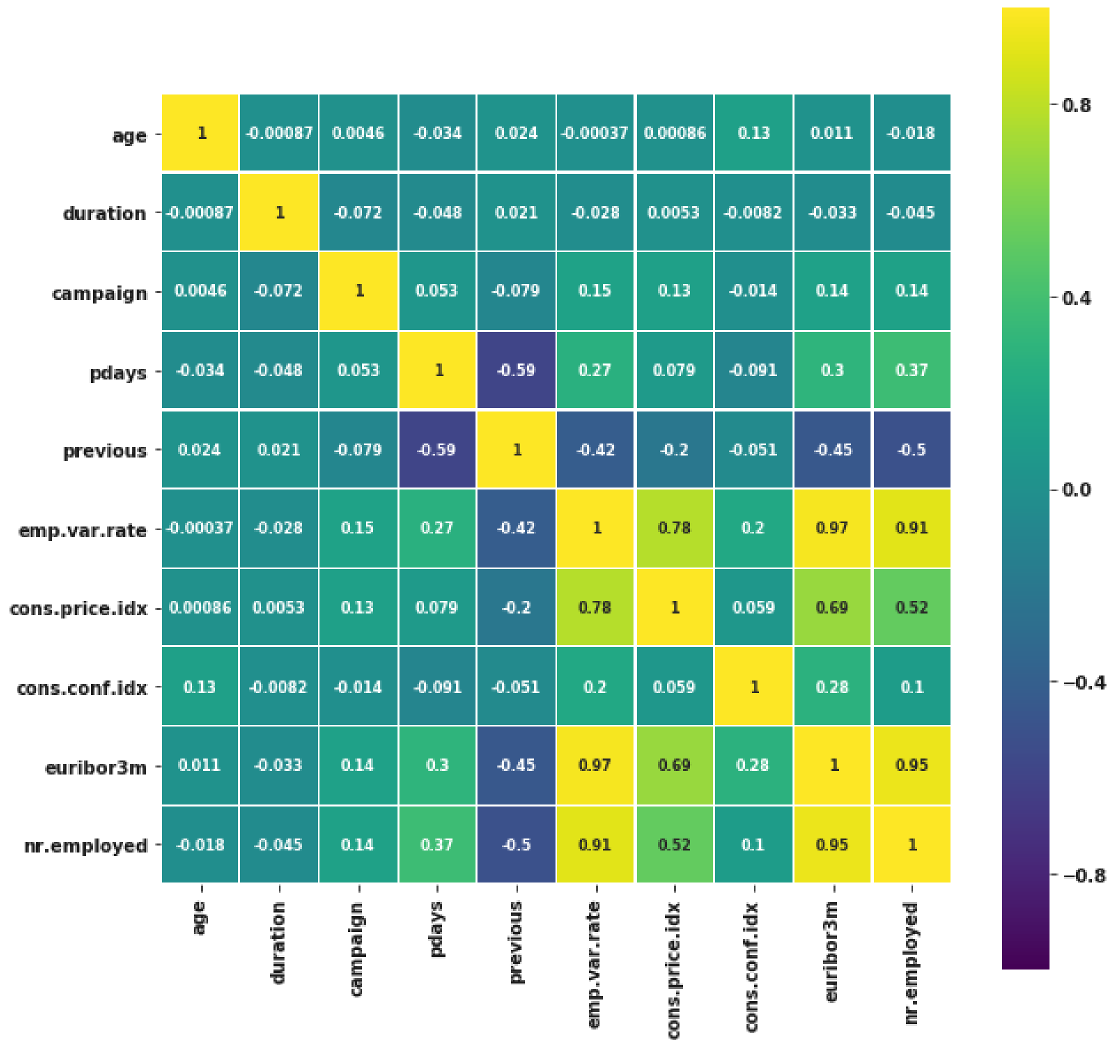
In [88]:
```
df_num = df_x.select_dtypes(include=['int64']).copy()

df_num.hist(figsize=(14, 10), bins=20, xlabelsize=8);
```



Numerical columns except **age** and **day** are positively skewed

duration is dropped as it is highly corerlated to the outcome as suggested in the dataset description.

```
In [89]: df_num = df_x.select_dtypes(include=['int64', 'float64']).copy()

         corr = df_num.corr()

         plt.figure(figsize=(10, 10))
         sns.heatmap(corr,
                     cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
                     annot=True, annot_kws={"size": 8}, square=True);
```

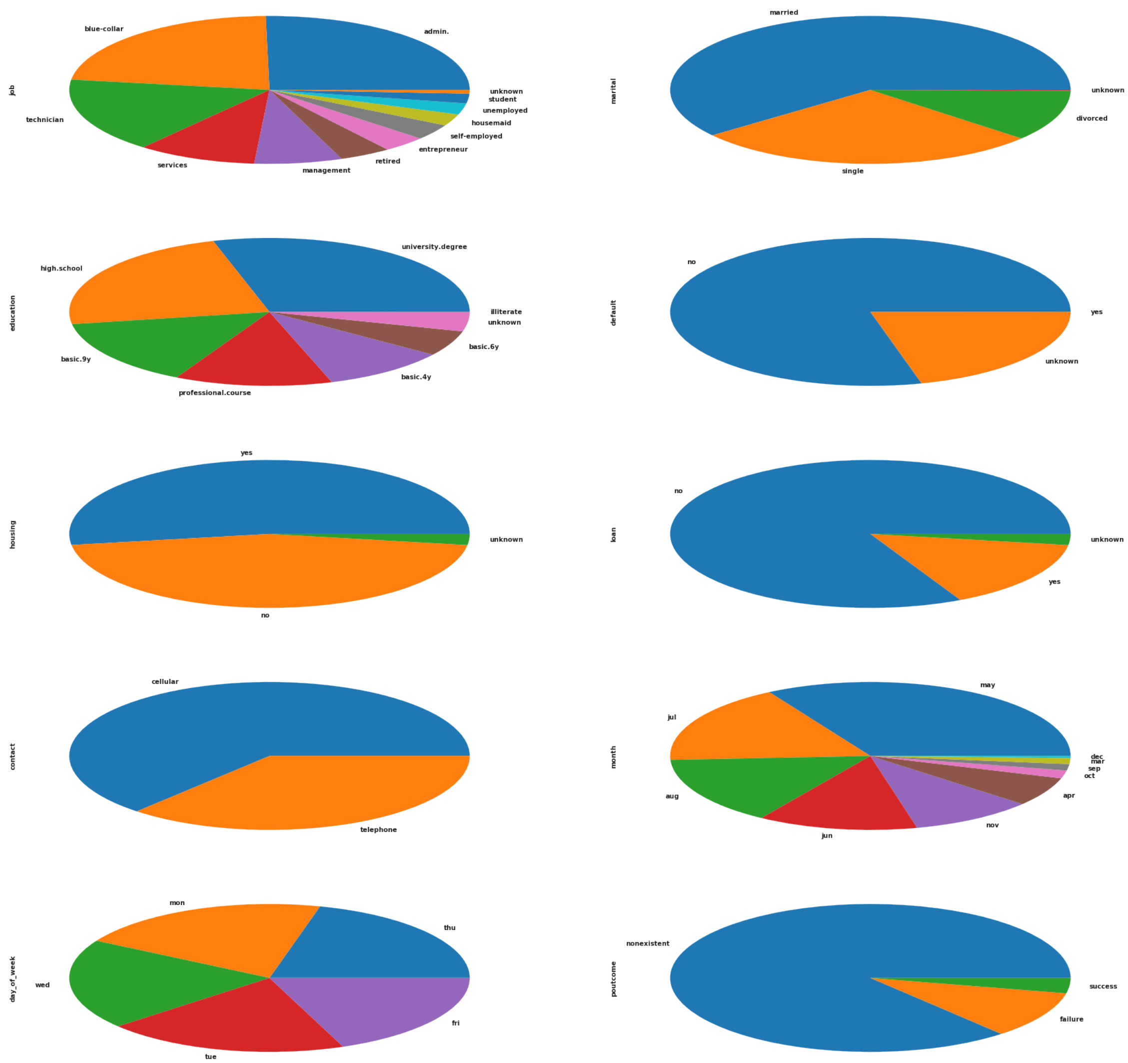


- **emp.var.rate** , **euribor3m** and **nr.employed** columns are highly correlated
- **pdays** and **previous** columns are negatively correlated

---

Visualising the distribution of the categorical attributes

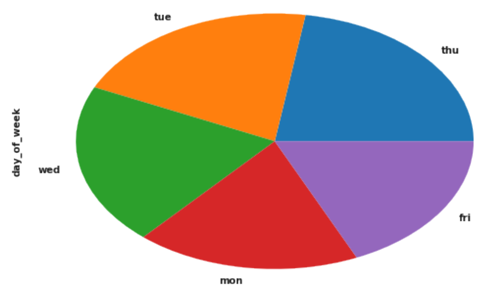
In [90]:
```
plt.figure(figsize=(30, 30))

for index, col in enumerate(df_x[categorical_attrs]):
    plt.subplot(5, 2, index+1)
    df[col].value_counts().plot.pie()
```

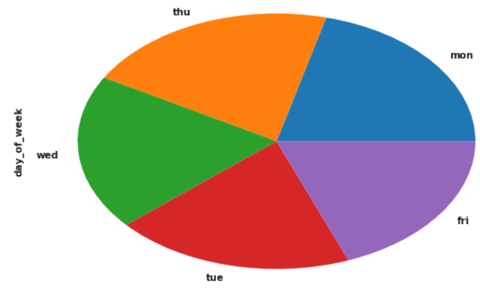


In [91]:
```
df_x.drop(['duration'], axis=1);
```

In [92]:
```
plt.figure(figsize=(20, 6));

plt.subplot(1, 2, 1)
df[df['y'] == 'yes']['day_of_week'].value_counts().plot.pie()

plt.subplot(1, 2, 2)
df[df['y'] == 'no']['day_of_week'].value_counts().plot.pie()
```

Out[92]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4d529cc88>
```



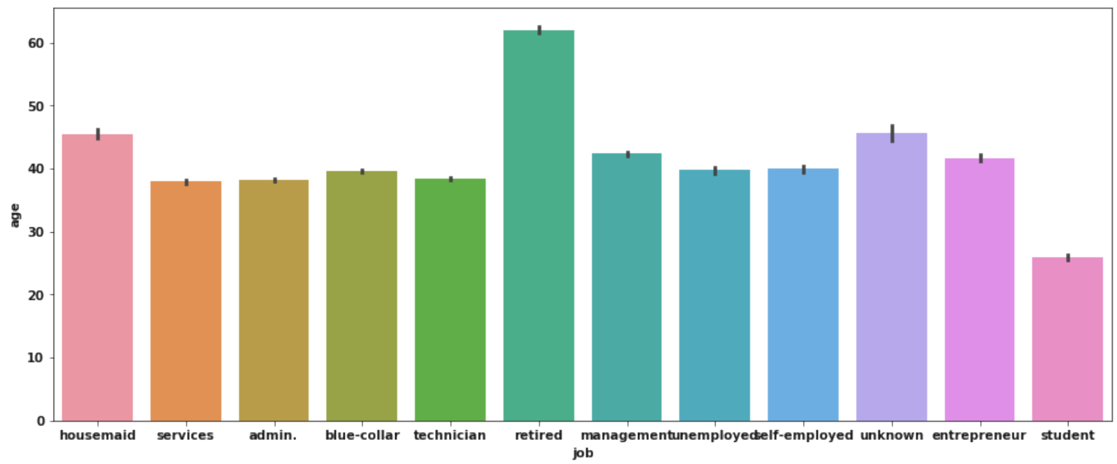


The **day_of_week** attributes are equally distributed and hence won't aid in classification

```
In [93]: df_x.drop(['day_of_week'], axis=1);
```

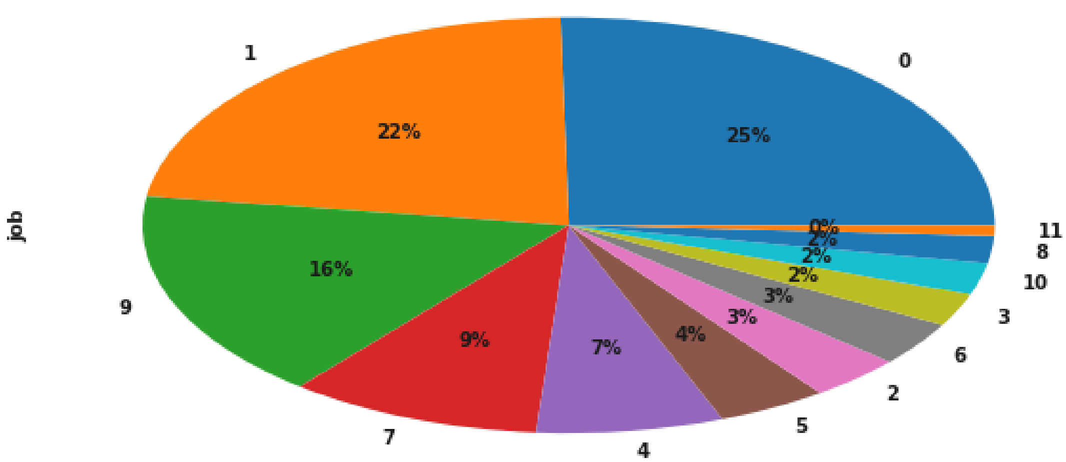
Age

```
In [94]: plt.figure(figsize=(15, 6))
         '''
         for index, col in enumerate(df_cat.columns):
             plt.subplot(5, 2, index+1)
             #plt.subplots_adjust(hspace=1)
             sns.barplot(x=col, y='age', hue='y', data=df)#, estimator=lambda x: len(x) / len(
             #if(index==3): break
         '''
         sns.barplot(x='job', y='age', data=df)

Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4d844fa20>
```



Retired people are having higher median age
Job

```
In [95]: df_x['job'].value_counts().plot.pie(autopct='%d%%', figsize=(10,5));
```



Blue collar and **management jobs** are the occupations that are majority

---

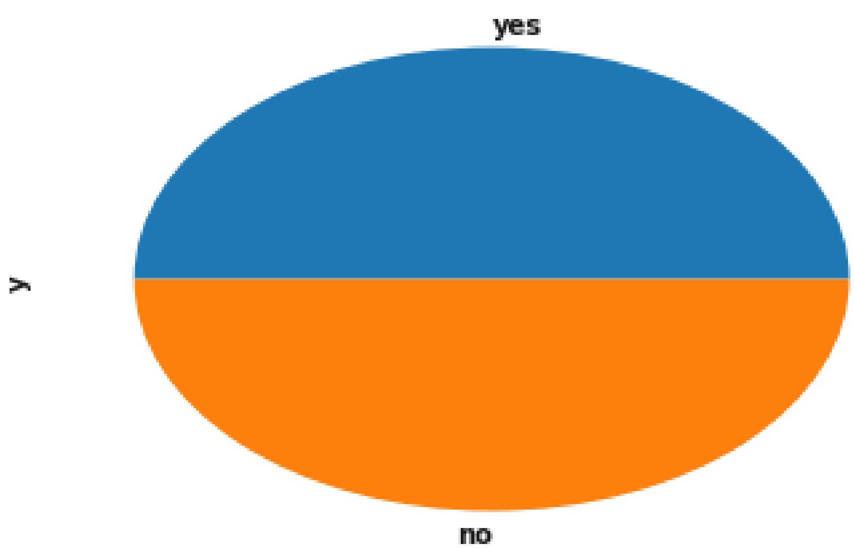
Preprocessing

```
In [96]: from sklearn.preprocessing import StandardScaler

         X = df_x.values
         y = df_y.values

         scaler = StandardScaler()
         stdScaledX = scaler.fit_transform(X)
         # summarize transformed data
         np.set_printoptions(precision=3)
         print(stdScaledX[0:2,:])
```

```
[[ 1.533 -0.202 -0.284 -1.754 -0.514 -1.088 -0.452  1.318  0.763 -0.719
   0.01  -0.566  0.195 -0.349  0.193  0.648  0.723  0.886  0.712  0.332]
 [ 1.629  0.911 -0.284 -0.35   1.945 -1.088 -0.452  1.318  0.763 -0.719
  -0.422 -0.566  0.195 -0.349  0.193  0.648  0.723  0.886  0.712  0.332]]
```

Checking class imbalance

```
In [97]: # number of samples who took term subscription
         num_yes = len(df_y[df_y['y']==1])

         # indices of samples that didnt take term subscription
         no_indices = df_y[df_y['y'] == 0].index

         # Random sample non term subscription
         random_indices = np.random.choice(no_indices,num_yes, replace=False)

         # Find the indices of term subscription
         yes_indices = df_y[df_y['y']==1].index

         # Concat yes indices with sample non ones
         under_sample_indices = np.concatenate([yes_indices,random_indices])

         # Get Balance Dataframe
         under_sample = df.loc[under_sample_indices]

         under_sample['y'].value_counts().plot.pie()

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4d64fab38>
```




```
In [98]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state
```

1.0.1 Find optimal parameters for the classifiers

1. Logistic Regression

```
In [99]: parameters = {'class_weight':['balanced', None]}

         clf =GridSearchCV(LogisticRegression(random_state=42), parameters)
         clf.fit(X_train, y_train)

         print("Best parameters are: {}".format(clf.best_params_))
         print(clf.cv_results_['mean_fit_time'])
         print(clf.cv_results_['mean_test_score'])
```

```
Best parameters are: {'class_weight': None}
[1.005 0.382]
[0.854 0.908]
```

---

2. SVM

```
In [100]: parameters = {'C':[1, 5, 10], 'class_weight':['balanced', None]}

          clf =GridSearchCV(LinearSVC(random_state=42), parameters)
          clf.fit(X_train, y_train)

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

```
Best parameters are: {'C': 1, 'class_weight': 'balanced'}
```

Multi Layer Perceptron

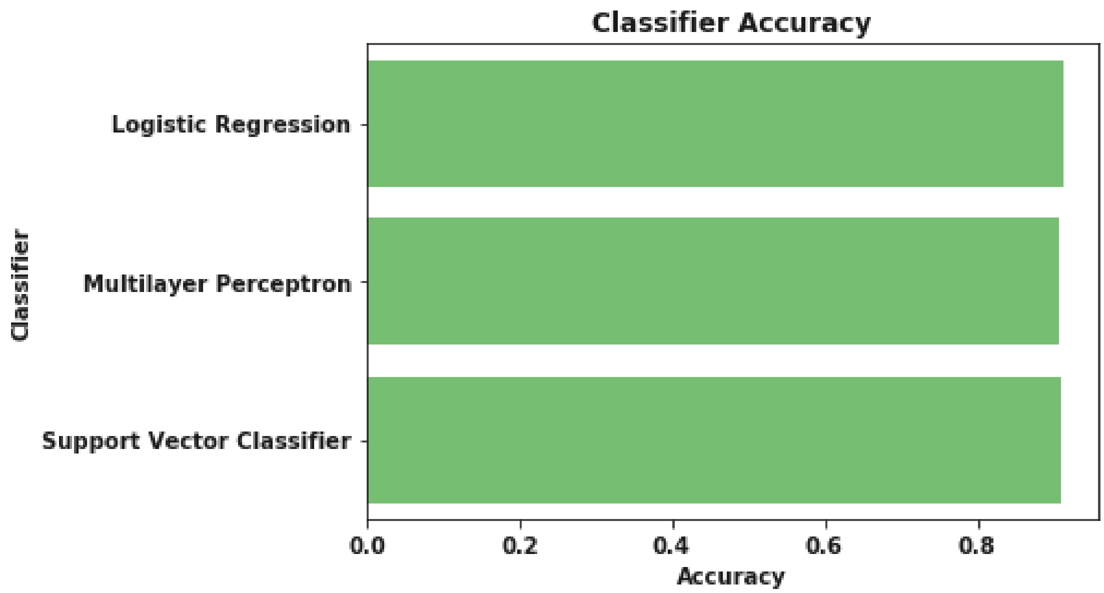
```
In [101]: parameters = {'activation':['tanh', 'relu'], 'hidden_layer_sizes':[(100,) , (100, 10

          clf =GridSearchCV(MLPClassifier(random_state=42), parameters)
          clf.fit(X_train, y_train)

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

```
Best parameters are: {'activation': 'tanh', 'hidden_layer_sizes': (100, 100)}
```

Classify with un-normalised features

```
In [102]: classifiers = {'Logistic Regression':LogisticRegression(random_state=42),
                         'Multilayer Perceptron': MLPClassifier(activation='tanh', hidden_laye
                         'Support Vector Classifier': LinearSVC(random_state=42)
                        }
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_sta

          classify_and_compare(classifiers, X_train, y_train, X_test, y_test)
```

```
Logistic Regression
             precision    recall  f1-score   support

          0       0.93      0.98      0.95      9144
          1       0.68      0.42      0.52      1153

avg / total       0.90      0.91      0.90     10297

Multilayer Perceptron
             precision    recall  f1-score   support

          0       0.93      0.97      0.95      9144
          1       0.62      0.43      0.51      1153

avg / total       0.90      0.91      0.90     10297

Support Vector Classifier
             precision    recall  f1-score   support

          0       0.93      0.97      0.95      9144
          1       0.64      0.47      0.54      1153

avg / total       0.90      0.91      0.90     10297

                  Classifier  Accuracy  Precision Score  Recall Score  \
0        Logistic Regression  0.912402         0.803615      0.695252
0      Multilayer Perceptron  0.907255         0.777226      0.700691
0  Support Vector Classifier  0.910654         0.786573      0.716627

   F1-Score  roc-auc_Score
0  0.733453       0.695252
0  0.730383       0.700691
0  0.744805       0.716627
```



Using standard scaled and undersampled dataset from now on

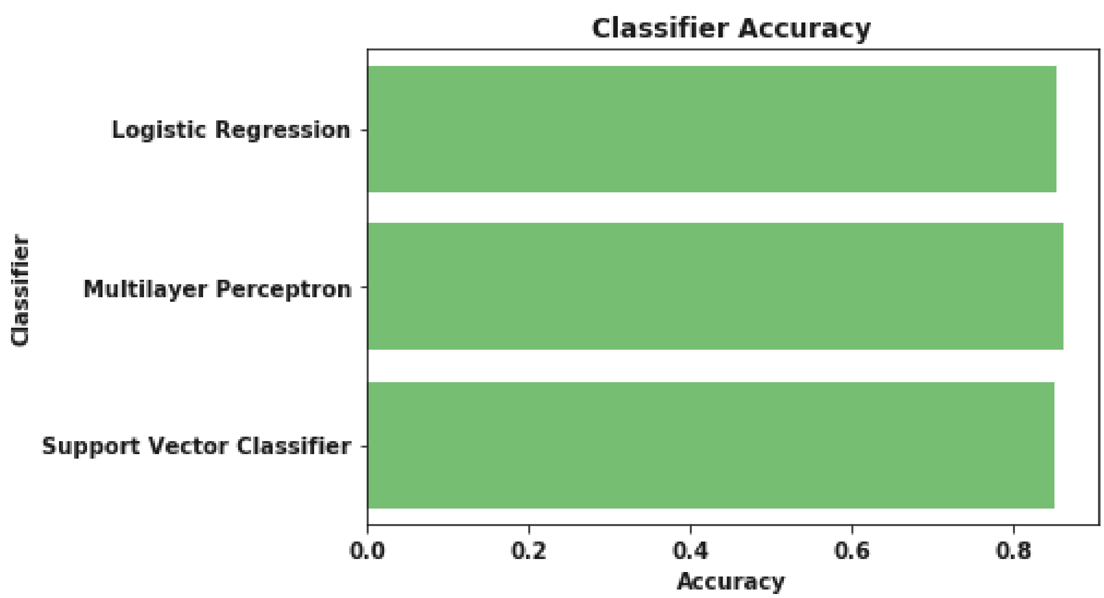
```
In [103]: normX = stdScaledX[under_sample_indices]
          normy = y[under_sample_indices]
```

Classification with normalised features

```
In [104]: classifiers = {'Logistic Regression':LogisticRegression(random_state=42),
                         'Multilayer Perceptron': MLPClassifier(activation='tanh', hidden_laye
                         'Support Vector Classifier': LinearSVC(random_state=42)
                        }
          X_train, X_test, y_train, y_test = train_test_split(normX,normy, test_size=0.25, ranc

          classify_and_compare(classifiers, X_train, y_train, X_test, y_test)
```

```
Logistic Regression
             precision    recall  f1-score   support

          0       0.87      0.84      0.85      1161
          1       0.84      0.87      0.86      1159

avg / total       0.85      0.85      0.85      2320

Multilayer Perceptron
             precision    recall  f1-score   support
```

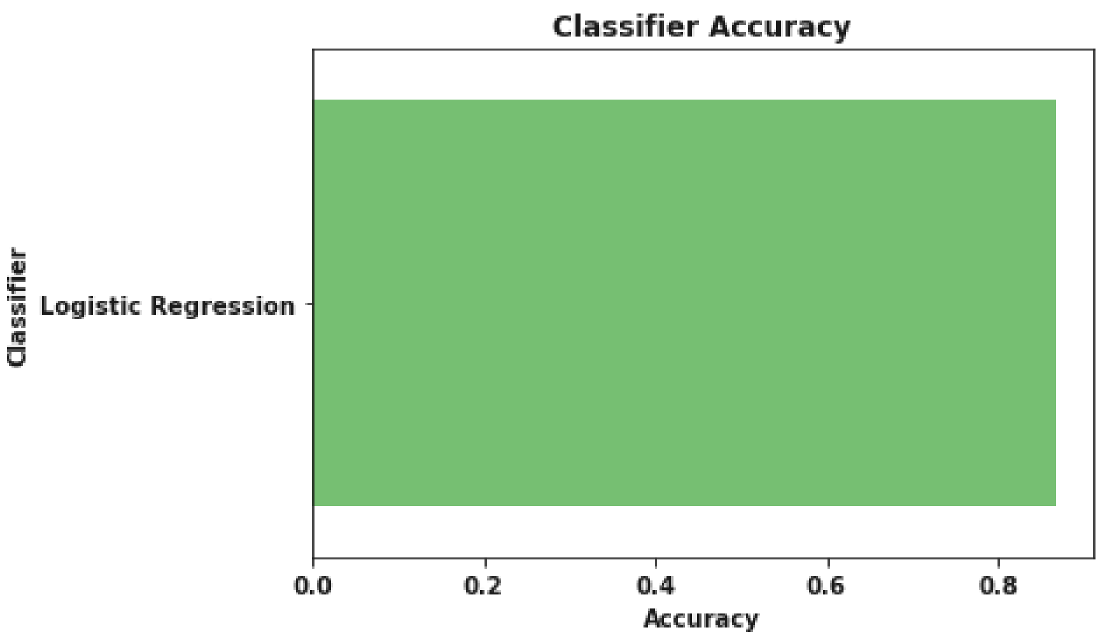
```
          0       0.87      0.85      0.86      1161
          1       0.85      0.88      0.86      1159

avg / total       0.86      0.86      0.86      2320

Support Vector Classifier
             precision    recall  f1-score   support

          0       0.86      0.84      0.85      1161
          1       0.84      0.86      0.85      1159

avg / total       0.85      0.85      0.85      2320

                  Classifier  Accuracy  Precision Score  Recall Score  \
0        Logistic Regression  0.853879         0.854234      0.853893
0      Multilayer Perceptron  0.861638         0.861962      0.861651
0  Support Vector Classifier  0.850862         0.851124      0.850874

   F1-Score  roc-auc_Score
0  0.853846       0.853893
0  0.861610       0.861651
0  0.850837       0.850874
```



Classification with polynomial features

```
In [105]: from sklearn.preprocessing import PolynomialFeatures
          poly = PolynomialFeatures()

          X_train, X_test, y_train, y_test = train_test_split(normX, normy, test_size=0.25, ra

          X_train = poly.fit_transform(X_train)
          #print(len(poly.get_feature_names()))
          #print(poly.get_params())

          X_test = poly.transform(X_test)

          classifiers = {'Logistic Regression':LogisticRegression(random_state=42),
                        }

          classify_and_compare(classifiers, X_train, y_train, X_test, y_test)
```

```
Logistic Regression
             precision    recall  f1-score   support

          0       0.89      0.84      0.86      1161
          1       0.85      0.89      0.87      1159

avg / total       0.87      0.87      0.87      2320

            Classifier  Accuracy  Precision Score  Recall Score  F1-Score  \
0  Logistic Regression   0.86681         0.867627      0.866831  0.866741

   roc-auc_Score
0       0.866831
```



1.0.2 PCA

In [106]:
```
from sklearn.decomposition import PCA

X_train, X_test, y_train, y_test = train_test_split(normX, normy, test_size=0.25, ran

# Make an instance of the Model
# 80% varience is retained
pca = PCA(.80)
pca.fit(X_train)

X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

print(X_train.shape)
```

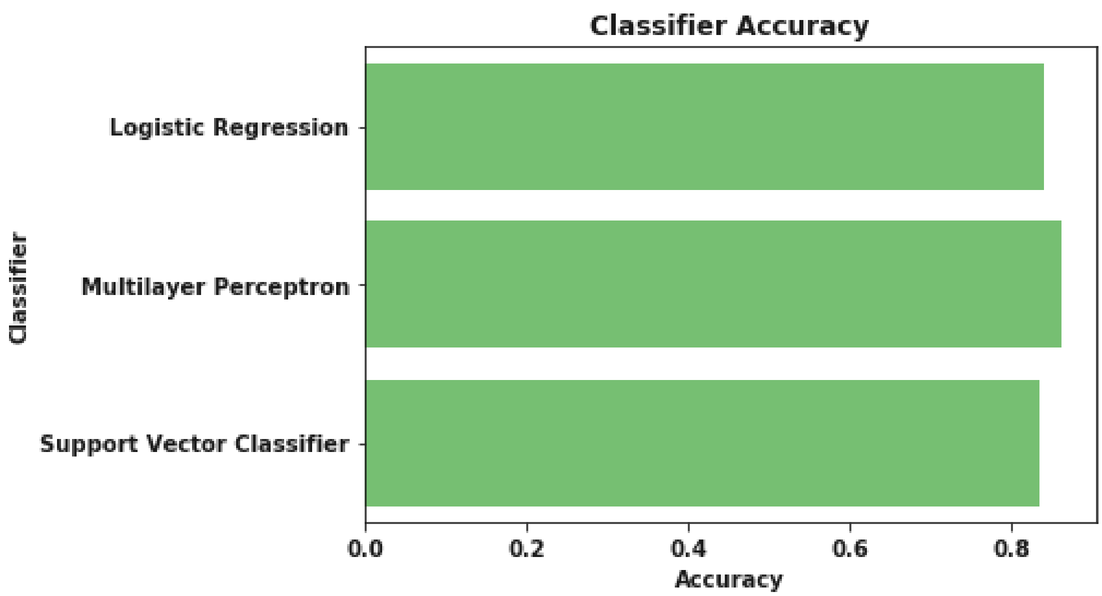
```
(6960, 10)
```

Classification after dimensionality reduction using PCA

In [107]:
```
classifiers = {'Logistic Regression':LogisticRegression(random_state=42),
               'Multilayer Perceptron': MLPClassifier(activation='tanh', hidden_laye
               'Support Vector Classifier': LinearSVC(random_state=42)
              }

classify_and_compare(classifiers, X_train, y_train, X_test, y_test)
```

```
Logistic Regression
             precision    recall  f1-score   support

          0       0.84      0.84      0.84      1161
          1       0.84      0.84      0.84      1159

avg / total       0.84      0.84      0.84      2320

Multilayer Perceptron
             precision    recall  f1-score   support

          0       0.88      0.83      0.86      1161
          1       0.84      0.89      0.87      1159

avg / total       0.86      0.86      0.86      2320

Support Vector Classifier
             precision    recall  f1-score   support
```

```
          0       0.83      0.84      0.84      1161
          1       0.84      0.83      0.83      1159

avg / total       0.84      0.84      0.84      2320

                   Classifier  Accuracy  Precision Score  Recall Score  \
0        Logistic Regression  0.840948         0.840955      0.840946
0      Multilayer Perceptron  0.862069         0.863164      0.862093
0  Support Vector Classifier  0.835345         0.835382      0.835340

   F1-Score  roc-auc_Score
0  0.840947       0.840946
0  0.861970       0.862093
0  0.835339       0.835340
```



PCA with polynomial features

In [108]:
```
X_train, X_test, y_train, y_test = train_test_split(normX, normy, test_size=0.25, ran

X_train = poly.fit_transform(X_train)
X_test = poly.transform(X_test)

print(len(poly.get_feature_names()))
```

```python
pca = PCA(.80)
pca.fit(X_train)

X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

print(X_train.shape)

classifiers = {'Logistic Regression':LogisticRegression(random_state=42),
               'Multilayer Perceptron': MLPClassifier(activation='tanh', hidden_layer
               'Support Vector Classifier': LinearSVC(random_state=42)
              }

classify_and_compare(classifiers, X_train, y_train, X_test, y_test)
```

```
231
(6960, 33)
Logistic Regression
             precision    recall  f1-score   support

          0       0.80      0.83      0.81      1161
          1       0.82      0.79      0.80      1159

avg / total       0.81      0.81      0.81      2320

Multilayer Perceptron
             precision    recall  f1-score   support

          0       0.84      0.84      0.84      1161
          1       0.84      0.84      0.84      1159

avg / total       0.84      0.84      0.84      2320

Support Vector Classifier
             precision    recall  f1-score   support

          0       0.79      0.84      0.81      1161
          1       0.83      0.77      0.80      1159

avg / total       0.81      0.81      0.81      2320

                  Classifier  Accuracy  Precision Score  Recall Score  \
0        Logistic Regression  0.809052         0.809753      0.809031
0      Multilayer Perceptron  0.837931         0.837931      0.837931
0  Support Vector Classifier  0.805603         0.806758      0.805577

   F1-Score  roc-auc_Score
0  0.808936       0.809031
```

```
0   0.837931          0.837931
0   0.805411          0.805577
```