```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "collapsed": true
   },
   "source": [
    "# Customer Churn Analysis"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "This notebook is using customer churn data from Kaggle (
https://www.kaggle.com/sandipdatta/customer-churn-analysis) and has been adopted from
the notebook available on Kaggle developed by SanD."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The notebook will go through the following steps:\n",
    "    1. Import Dataset\n",
    "    2. Analyze the Data\n",
    "    3. Prepare the data model building\n",
    "    4. Split data in test and train data\n",
    "    5. Train model using various machine learning algorithms for binary
classification\n",
    "    6. Evaluate the models\n",
    "    7. Select the model best fit for the given data set\n",
    "    8. Save and deploy model to Watson Machine Learning"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn import model_selection\n",
    "from sklearn import tree\n",
    "from sklearn import svm\n",
    "from sklearn import ensemble\n",
    "from sklearn import neighbors\n",
    "from sklearn import linear_model\n",
    "from sklearn import metrics\n",
    "from sklearn import preprocessing"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "%matplotlib inline \n",
    "\n",
    "from IPython.display import Image\n",
    "import matplotlib as mlp\n",
    "import matplotlib.pyplot as plt\n",
    "import numpy as np\n",
    "import os\n",
    "import pandas as pd\n",
```

```
    "import sklearn\n",
    "import seaborn as sns\n",
    "import json"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Dataset"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The original dataset can be downloaded from
https://www.kaggle.com/becksddf/churn-in-telecoms-dataset/data. Then upload it to IBM
Watson Studio and insert the code to read the data using \"insert to code > Insert
panndas DataFrame\"."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# @hidden_cell\n",
    "\n",
    "# make sure you assign the dataframe to the variable \"df\"\n",
    "df = df_data_X\n",
    "print (df.shape)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Examine the first 5 lines of the input"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "df.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "y = df[\"churn\"].value_counts()\n",
    "sns.barplot(y.index, y.values)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
```

```
   "source": [
    "y_True = df[\"churn\"][df[\"churn\"] == True]\n",
    "print (\"Churn Percentage = \"+str( (y_True.shape[0] / df[\"churn\"].shape[0]) *
100 ))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Descriptive Analysis of the Data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "  df.describe()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Churn by State "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "df.groupby([\"state\", \"churn\"]).size().unstack().plot(kind='bar',
stacked=True, figsize=(30,10)) "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Churn by Area Code "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "df.groupby([\"area code\", \"churn\"]).size().unstack().plot(kind='bar',
stacked=True, figsize=(5,5)) "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "###  Churn by customers with International Plan "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
```

```
   "metadata": {},
   "outputs": [],
   "source": [
    "df.groupby([\"international plan\", \"churn\"]).size().unstack().plot(kind='bar',
stacked=True, figsize=(5,5)) "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Churn By Customers with Voice mail plan"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "df.groupby([\"voice mail plan\", \"churn\"]).size().unstack().plot(kind='bar',
stacked=True, figsize=(5,5)) "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Data Preparation"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The following preprocessing steps need to be done:\n",
    "1. Turn categorical variables into discrete numerical variables\n",
    "2. Create response vector\n",
    "3. Drop superflous columns\n",
    "4. Build feature matrix\n",
    "5. Standardize feature matrix values"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Encode categorical columns"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# Discreet value integer encoder\n",
    "label_encoder = preprocessing.LabelEncoder()\n",
    "\n",
    "# State, international plans and voice mail plan are strings and we want discreet
integer values\n",
    "df['state'] = label_encoder.fit_transform(df['state'])\n",
    "df['international plan'] = label_encoder.fit_transform(df['international
plan'])\n",
    "df['voice mail plan'] = label_encoder.fit_transform(df['voice mail plan'])\n",
    "\n",
```

```
   "print (df.dtypes)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [],
  "source": [
   "print (df.shape)\n",
   "df.head()"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "### Create response vector"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "y = df['churn'].values.astype(np.int)\n",
   "y.size"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "### Drop superflous columns"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "# df = df.drop([\"Id\",\"Churn\"], axis = 1, inplace=True)\n",
   "df.drop([\"phone number\",\"churn\"], axis = 1, inplace=True)\n",
   "df.head()"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "### Build feature matrix"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "X = df.values.astype(np.float)\n",
   "print(X)\n",
   "X.shape"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Standardize Feature Matrix values"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "scaler = preprocessing.StandardScaler()\n",
    "X = scaler.fit_transform(X)\n",
    "X"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "This completes the data preparation steps."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Split Train/Test Validation Data"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "We need to adopt Stratified Cross Validation - Since the Response values are not
balanced"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "def stratified_cv(X, y, clf_class, shuffle=True, n_folds=10):\n",
    "    stratified_k_fold = model_selection.StratifiedKFold(n_splits=n_folds,
shuffle=shuffle)\n",
    "    y_pred = y.copy()\n",
    "    # ii -> train\n",
    "    # jj -> test indices\n",
    "    for ii, jj in stratified_k_fold.split(X, y): \n",
    "        X_train, X_test = X[ii], X[jj]\n",
    "        y_train = y[ii]\n",
    "        clf = clf_class\n",
    "        clf.fit(X_train,y_train)\n",
    "        y_pred[jj] = clf.predict(X_test)\n",
    "    return y_pred"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
```

```
   "source": [
    "## Build Models and Train"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "We will build models using a variety of approaches to see how they compare:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# create classifiers\n",
    "from sklearn.ensemble import GradientBoostingClassifier\n",
    "gradient_boost = GradientBoostingClassifier()\n",
    "\n",
    "from sklearn.svm import SVC\n",
    "svc_model = SVC(gamma='auto')\n",
    "\n",
    "from sklearn.ensemble import RandomForestClassifier\n",
    "random_forest = RandomForestClassifier(n_estimators=10)\n",
    "\n",
    "from sklearn.neighbors import KNeighborsClassifier\n",
    "k_neighbors = KNeighborsClassifier()\n",
    "\n",
    "from sklearn.linear_model import LogisticRegression\n",
    "logistic_regression = LogisticRegression(solver='lbfgs')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "print('Gradient Boosting Classifier:  {:.2f}'.format(metrics.accuracy_score(y,
stratified_cv(X, y, gradient_boost))))\n",
    "print('Support vector machine(SVM):   {:.2f}'.format(metrics.accuracy_score(y,
stratified_cv(X, y, svc_model))))\n",
    "print('Random Forest Classifier:      {:.2f}'.format(metrics.accuracy_score(y,
stratified_cv(X, y, random_forest))))\n",
    "print('K Nearest Neighbor Classifier: {:.2f}'.format(metrics.accuracy_score(y,
stratified_cv(X, y, k_neighbors))))\n",
    "print('Logistic Regression:           {:.2f}'.format(metrics.accuracy_score(y,
stratified_cv(X, y, logistic_regression))))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Model Evaluation"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "We will now generate confusion matrices for the various models to analyze the
prediction in more detail."
   ]
```

```
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Gradient Boosting Classifier"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "grad_ens_conf_matrix = metrics.confusion_matrix(y, stratified_cv(X, y,
gradient_boost))\n",
    "sns.heatmap(grad_ens_conf_matrix, annot=True,  fmt='');\n",
    "title = 'Gradient Boosting'\n",
    "plt.title(title);"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Support Vector Machines"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "svm_svc_conf_matrix = metrics.confusion_matrix(y, stratified_cv(X, y,
svc_model))\n",
    "sns.heatmap(svm_svc_conf_matrix, annot=True,  fmt='');\n",
    "title = 'SVM'\n",
    "plt.title(title);"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Random Forest"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "random_forest_conf_matrix = metrics.confusion_matrix(y, stratified_cv(X, y,
random_forest))\n",
    "sns.heatmap(random_forest_conf_matrix, annot=True,  fmt='');\n",
    "title = 'Random Forest'\n",
    "plt.title(title);"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Classification Report"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "print('Gradient Boosting Classifier:\\n
{}\\n'.format(metrics.classification_report(y, stratified_cv(X, y,
gradient_boost))))\n",
    "print('Support vector machine(SVM):\\n
{}\\n'.format(metrics.classification_report(y, stratified_cv(X, y, svc_model))))\n",
    "print('Random Forest Classifier:\\n
{}\\n'.format(metrics.classification_report(y, stratified_cv(X, y, random_forest))))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Final Model Selection"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Gradient Boosting seems to do comparatively better for this case"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "gbc = ensemble.GradientBoostingClassifier()\n",
    "gbc.fit(X, y)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "# Get Feature Importance from the classifier\n",
    "feature_importance = gbc.feature_importances_\n",
    "print (gbc.feature_importances_)\n",
    "feat_importances = pd.Series(gbc.feature_importances_, index=df.columns)\n",
    "feat_importances = feat_importances.nlargest(19)\n",
    "feat_importances.plot(kind='barh' , figsize=(10,10)) "
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Save and Deploy model to Watson Machine Learning"
   ]
  },
  {
   "cell_type": "markdown",
```

```
   "metadata": {},
   "source": [
    "### Connection to WML\n",
    "\n",
    "To authenticate the Watson Machine Learning service on IBM Cloud, you will need
to provide a platform `api_key` and instance `location`.\n",
    "\n",
    "You can use the [IBM Cloud CLI](https://cloud.ibm.com/docs/cli/index.html) or IBM
Cloud console to create your API key.\n",
    "\n",
    "Using the IBM Cloud CLI:\n",
    "\n",
    "```bash\n",
    "ibmcloud login\n",
    "ibmcloud iam api-key-create API_KEY_NAME\n",
    "```\n",
    "\n",
    "Retrieve the value of api_key from the output.\n",
    "\n",
    "```bash\n",
    "ibmcloud login --apikey API_KEY -a https://cloud.ibm.com\n",
    "ibmcloud resource service-instance WML_INSTANCE_NAME\n",
    "```\n",
    "\n",
    "Retrieve the value of location from the output.\n",
    "\n",
    "Using the IBM Cloud console:\n",
    "\n",
    "Navigate to the [Users panel](https://cloud.ibm.com/iam#/users). Then click your
name, scroll down to the **API Keys** section, and click **Create an IBM Cloud API
key**. Give your key a name and click **Create**, then copy the created key and paste
it below. You can retrieve your instance location in your [Watson Machine Learning
(WML) Service](https://console.ng.bluemix.net/catalog/services/ibm-watson-machine-
learning/) instance details.\n",
    "\n",
    "You can also get service specific apikey by going to the [Service IDs section of
the Cloud Console](https://cloud.ibm.com/iam/serviceids). From that page, click
**Create**, then copy the created key and paste it below.\n",
    "\n",
    "**NOTE**: You can also get a service specific url. Go to the [Endpoint URLs
section of the Watson Machine Learning docs](https://cloud.ibm.com/apidocs/machine-
learning) for details."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "api_key = 'PASTE YOUR PLATFORM API KEY HERE'\n",
    "location = 'PASTE YOUR INSTANCE LOCATION HERE'"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "wml_credentials = {\n",
    "    \"apikey\": api_key,\n",
    "    \"url\": 'https://' + location + '.ml.cloud.ibm.com'\n",
    "}"
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Install and import the ibm-watson-machine-learning package\n",
    "\n",
    "Note: ibm-watson-machine-learning documentation can be found [here](http://ibm-wml-api-pyclient.mybluemix.net/)."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "!pip install -U ibm-watson-machine-learning"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# create client to access our WML service\n",
    "from ibm_watson_machine_learning import APIClient\n",
    "\n",
    "client = APIClient(wml_credentials)\n",
    "print(client.version)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Working with spaces\n",
    "\n",
    "First, create a space that will be used for your work. If you do not have space already created, you can use [Deployment Spaces dashboard](https://dataplatform.cloud.ibm.com/ml-runtime/spaces?context=cpdaas) to create one.\n",
    "\n",
    "* Click New Deployment Space\n",
    "* Create an empty space\n",
    "* Select Cloud Object Storage\n",
    "* Select Watson Machine Learning instance and press Create\n",
    "* Copy space_id and paste it below\n",
    "\n",
    "**Tip**: You can also use WML SDK to prepare the space for your work. More information can be found [here](https://github.com/IBM/watson-machine-learning-samples/blob/master/cloud/notebooks/python_sdk/instance-management/Space%20management.ipynb).\n",
    "\n",
    "**Action**: Assign space ID below"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "space_id = 'PASTE YOUR SPACE ID HERE'"
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "You can use list method to print all existing spaces."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "client.spaces.list(limit=10)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "To be able to interact with all resources available in Watson Machine Learning,
you need to set the **space** which you will be using."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "client.set.default_space(space_id)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Upload model\n",
    "\n",
    "In this section you will learn how to upload the model to the Cloud."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "sofware_spec_uid =
client.software_specifications.get_id_by_name(\"default_py3.7\")\n",
    "metadata = {\n",
    "            client.repository.ModelMetaNames.NAME: 'Gradient Boosting model to
predict customer churn',\n",
    "            client.repository.ModelMetaNames.TYPE: 'scikit-learn_0.23',\n",
    "            client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:
sofware_spec_uid\n",
    "}\n",
    "\n",
    "published_model = client.repository.store_model(\n",
    "    model=gbc,\n",
    "    meta_props=metadata)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
```

"source": [
 "Use the following command to get details about the model"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": [
 "# Get model details\n",
 "import json\n",
 "\n",
 "published_model_uid = client.repository.get_model_uid(published_model)\n",
 "model_details = client.repository.get_details(published_model_uid)\n",
 "print(json.dumps(model_details, indent=2))"
]
},
{
"cell_type": "markdown",
"metadata": {},
"source": [
 "Note: You can see that model is successfully stored in Watson Machine Learning
Service."
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": [
 "client.repository.list_models()"
]
},
{
"cell_type": "markdown",
"metadata": {},
"source": [
 "Use the following command to clean up/delete any previously created models"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": [
 "# client.repository.delete('GUID of stored model')"
]
},
{
"cell_type": "markdown",
"metadata": {},
"source": [
 "### Create online deployment\n",
 "\n",
 "You can use commands bellow to deploy the stored model as a web service."
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": [

```
    "# Create online deployment\n",
    "metadata = {\n",
    "    client.deployments.ConfigurationMetaNames.NAME: \"Deployment of customer
churn model\",\n",
    "    client.deployments.ConfigurationMetaNames.ONLINE: {}\n",
    "}\n",
    "\n",
    "created_deployment = client.deployments.create(published_model_uid,
meta_props=metadata)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Use the following commands to retrieve the deployment UID, show all deployments,
and to delete old deployments."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# Get deployment UID and show details on the deployment\n",
    "deployment_uid = client.deployments.get_uid(created_deployment)\n",
    "client.deployments.get_details(deployment_uid)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# list all deployments\n",
    "client.deployments.list()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# delete old deployments\n",
    "# client.deployments.delete('GUID of deployed model')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Scoring\n",
    "\n",
    "You can send new scoring records to the web-service deployment using the WML
**score** method."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
```

```
    "# get scoring end point\n",
    "scoring_endpoint = client.deployments.get_scoring_href(created_deployment)\n",
    "print(scoring_endpoint)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# use our WML client to score our model\n",
    "# add some test data\n",
    "scoring_payload = {\"input_data\": [\n",
    "    {'fields': ['state', 'account length', 'area code', 'international plan',
'voice mail plan', 'number vmail messages', 'total day minutes', \n",
    "                 'total day calls', 'total day charge', 'total eve minutes',
'total eve calls', 'total eve charge', 'total night minutes', \n",
    "                 'total night calls', 'total night charge', 'total intl minutes',
'total intl calls', 'total intl charge', 'customer service calls' ], \n",
    "     'values': [[
'2','162','415','0','0','0','70.7','108','12.02','157.5','87','13.39','154.8','82','6.
97','9.1','3','2.46','4' ]]\n",
    "    }]}"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# score the model\n",
    "predictions = client.deployments.score(deployment_uid, scoring_payload)\n",
    "print('prediction',json.dumps(predictions, indent=2))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Acknowledgement"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The approach and code fragments have been adopted from the nootebook on Kaggle by
Sandip Datta (https://www.kaggle.com/sandipdatta). \n",
    "The full original notebook can be viewed here:
https://www.kaggle.com/sandipdatta/customer-churn-analysis#"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3.7",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
```

```
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.7.9"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 1
}
```