Project1

February 26, 2020

Start by importing the packages we need

```
[ ]: import pandas as pd
     import numpy as np
     import os
     import sklearn
     import matplotlib
```

Next step is to read in the data

```
[2]: myData = pd.read_csv("~/Documents/Titanic/pp-complete.csv", header = None)
```

Lets check out the data to understand it better

```
[3]: myData.head()
     myData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25018011 entries, 0 to 25018010
Data columns (total 16 columns):
 #   Column  Dtype
---  ------  -----
 0   0       object
 1   1       int64
 2   2       object
 3   3       object
 4   4       object
 5   5       object
 6   6       object
 7   7       object
 8   8       object
 9   9       object
 10  10      object
 11  11      object
 12  12      object
 13  13      object
 14  14      object
 15  15      object
```

```
dtypes: int64(1), object(15)
memory usage: 3.0+ GB
```

Select variables of interest and rename them to make it more clear

[4]:

```
modelData = myData[[1,2,4,6,11]]
modelData = modelData.rename(columns={1: "Price", 2: "Date",4: "Type", 6:␣
 ↪"Duration", 11:"Location"})
modelData.info()
modelData.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25018011 entries, 0 to 25018010
Data columns (total 5 columns):
 #   Column    Dtype
---  ------    -----
 0   Price     int64
 1   Date      object
 2   Type      object
 3   Duration  object
 4   Location  object
dtypes: int64(1), object(4)
memory usage: 954.4+ MB
```

[4]:

```
   Price              Date Type Duration    Location
0  18500  1995-01-31 00:00    F        L    TORQUAY
1  73450  1995-10-09 00:00    D        F   LIVERPOOL
2  59000  1995-03-31 00:00    D        F       POOLE
3  31000  1995-12-04 00:00    D        F  WOODBRIDGE
4  95000  1995-09-22 00:00    D        F   LICHFIELD
```

Format the date column and get the year from the date which will be used to split the dataset later

[5]:

```
modelData['Date'] = pd.to_datetime(modelData['Date'], format = '%Y-%m-%d')

modelData['Year'] = modelData['Date'].dt.year

modelData.info()
modelData.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25018011 entries, 0 to 25018010
Data columns (total 6 columns):
 #   Column    Dtype
---  ------    -----
 0   Price     int64
 1   Date      datetime64[ns]
 2   Type      object
 3   Duration  object
 4   Location  object
 5   Year      int64
dtypes: datetime64[ns](1), int64(2), object(3)
memory usage: 1.1+ GB
```

[5]:

```
    Price       Date Type Duration     Location  Year
0   18500 1995-01-31    F        L      TORQUAY  1995
1   73450 1995-10-09    D        F    LIVERPOOL  1995
2   59000 1995-03-31    D        F        POOLE  1995
3   31000 1995-12-04    D        F   WOODBRIDGE  1995
4   95000 1995-09-22    D        F    LICHFIELD  1995
5   45450 1995-02-28    S        F  CHESTERFIELD  1995
6   96000 1995-10-27    S        F        EPSOM  1995
7   30000 1995-11-28    S        F   WEDNESBURY  1995
8  425000 1995-03-31    D        F       COBHAM  1995
9   89995 1995-06-30    D        F    NORMANTON  1995
```

Add dummy variable "one-hot encode variable" if location of house is in London, use DataFrame from pandas to do it

[6]:

```python
from pandas import DataFrame
modelData['isLondon'] = modelData['Location'].apply(lambda x: 1 if x ==␣
 ↪'LONDON' else 0)

modelData.info()
```

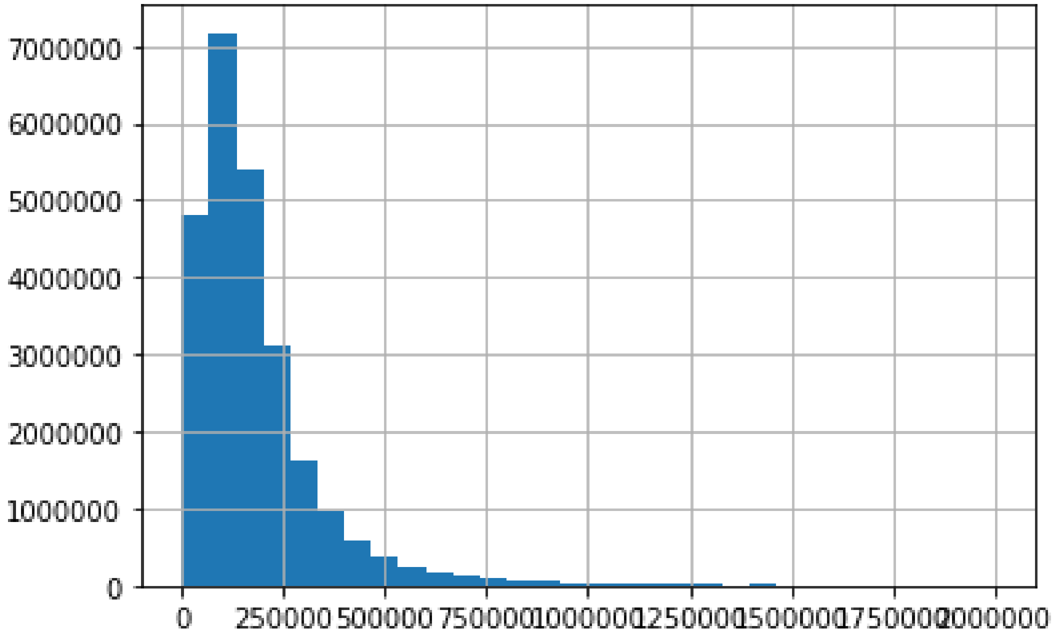
```
            Price       Date Type Duration    Location  Year  isLondon
0           18500 1995-01-31    F        L     TORQUAY  1995         0
1           73450 1995-10-09    D        F   LIVERPOOL  1995         0
2           59000 1995-03-31    D        F       POOLE  1995         0
3           31000 1995-12-04    D        F  WOODBRIDGE  1995         0
4           95000 1995-09-22    D        F   LICHFIELD  1995         0
…             …         …   …        …         …   …            …
25018006   410854 2019-07-18    D        F      HORLEY  2019         0
25018007   610000 2019-08-08    D        F    CATERHAM  2019         0
25018008    42500 2019-07-22    O        F   GUILDFORD  2019         0
25018009   353500 2019-08-02    O        F    CHERTSEY  2019         0
25018010  1185000 2019-08-09    D        F      SUTTON  2019         0

[25018011 rows x 7 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25018011 entries, 0 to 25018010
Data columns (total 7 columns):
 #   Column    Dtype
---  ------    -----
 0   Price     int64
 1   Date      datetime64[ns]
 2   Type      object
 3   Duration  object
 4   Location  object
 5   Year      int64
 6   isLondon  int64
dtypes: datetime64[ns](1), int64(3), object(3)
memory usage: 1.3+ GB
```

Make a histogram of the price to see the distribution, notice it has a heavy tail on the lef

[7]:

```
import matplotlib.pyplot as plt
%matplotlib inline
modelData['Price'].hist(bins=30, range=(0, 2000000))
plt.show()
```



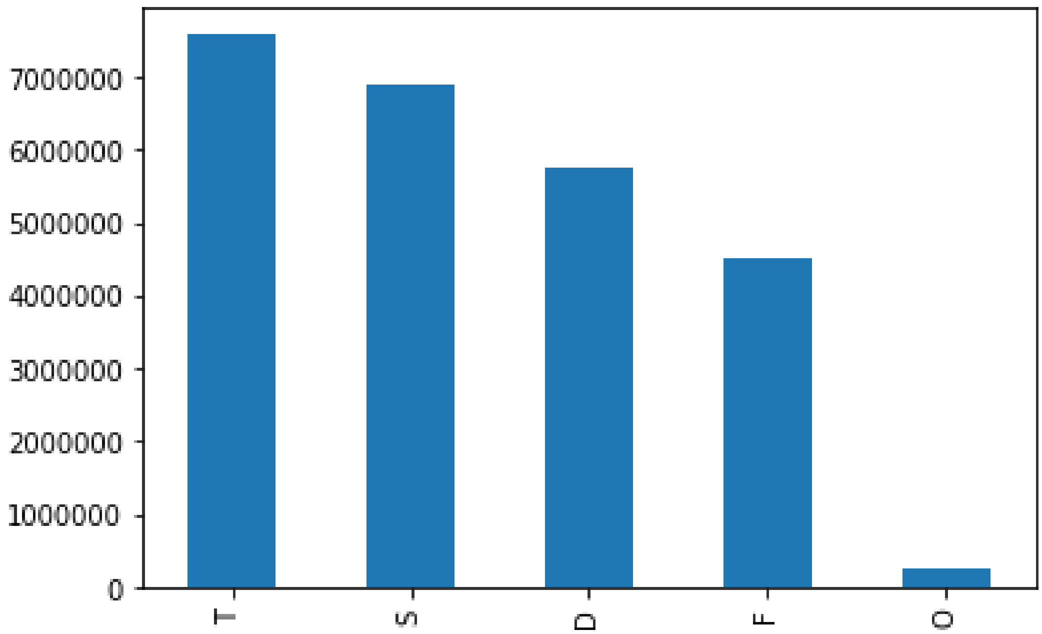
Make some bar charts for the categorical variables to see how common each category is

[8]:

```
#Get some bar charts of the categorical variables
modelData['Type'].value_counts().plot(kind='bar')
```

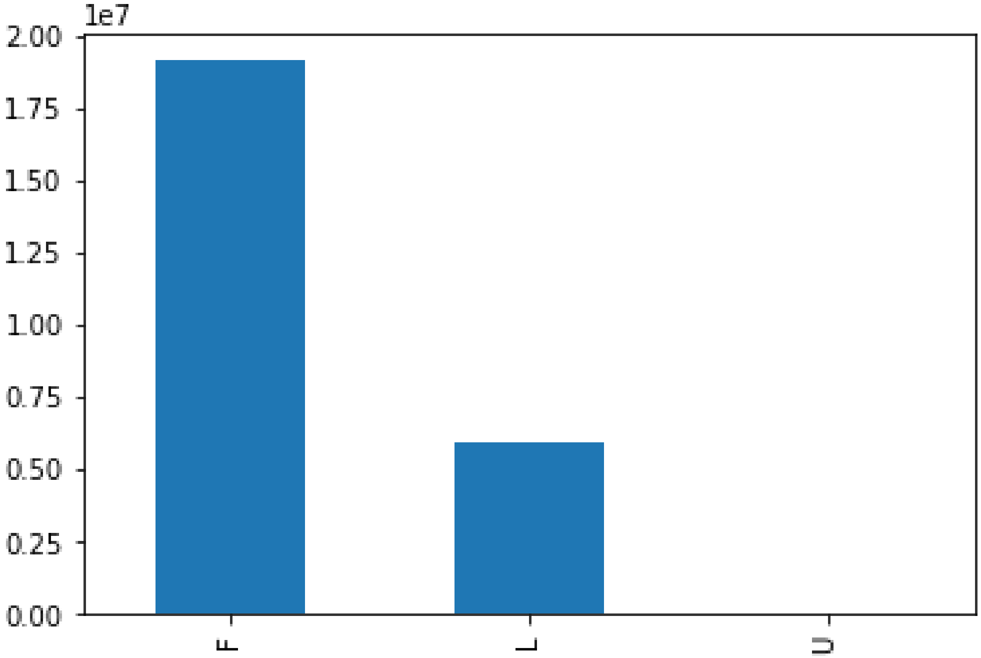
[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e03ad5dd0>
```



```
[9]: modelData['Duration'].value_counts().plot(kind='bar')
```

```
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e030f4f90>
```



Get dummy variables for each type of house and duration and add it to the dataset, rename them also to avoid confusion between variables

[10]:

```
typeDummy = pd.get_dummies(modelData['Type'])
modelData = modelData.drop('Type',axis = 1)
modelData = modelData.join(typeDummy)


modelData = modelData.rename(columns={'D': "TypeD",'F': "TypeF",'O': "TypeO",␣
 ↪'S': "TypeS", 'T': "TypeT"})
```

[11]:

```
durationDummy = pd.get_dummies(modelData['Duration'])
modelData = modelData.drop('Duration',axis = 1)
modelData = modelData.join(durationDummy)

modelData = modelData.rename(columns = {'F': 'DurationF', 'L': 'DurationL', 'U':
 ↪ 'DurationU'})
```

Make a copy of the dataset which we will later split to train and test sets

[12]:

```
cleanedData = modelData.copy()
cleanedData.info()
```

[13]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25018011 entries, 0 to 25018010
Data columns (total 13 columns):
 #   Column     Dtype
---  ------     -----
 0   Price      int64
 1   Date       datetime64[ns]
 2   Location   object
 3   Year       int64
 4   isLondon   int64
 5   TypeD      uint8
 6   TypeF      uint8
 7   TypeO      uint8
 8   TypeS      uint8
 9   TypeT      uint8
 10  DurationF  uint8
 11  DurationL  uint8
 12  DurationU  uint8
dtypes: datetime64[ns](1), int64(3), object(1), uint8(8)
memory usage: 1.1+ GB
```

Delete variables we will not use and split the dataset into a training and test set. The test set includes variables which were sold in 2015 while the training set includes all the other properties.

[14]:

```
del cleanedData['Date']
del cleanedData['Location']
```

[15]:

```
#Split to test and training set, test data is data in december, rest is training data
trainData = cleanedData[cleanedData.Year != 2015]
trainData.info()
testData = cleanedData[cleanedData.Year == 2015]
testData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24008337 entries, 0 to 25018010
Data columns (total 11 columns):
 #   Column     Dtype
---  ------     -----
 0   Price      int64
 1   Year       int64
 2   isLondon   int64
 3   TypeD      uint8
 4   TypeF      uint8
 5   TypeO      uint8
 6   TypeS      uint8
 7   TypeT      uint8
 8   DurationF  uint8
 9   DurationL  uint8
 10  DurationU  uint8
dtypes: int64(3), uint8(8)
memory usage: 915.8 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1009674 entries, 20008866 to 21018539
Data columns (total 11 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   Price      1009674 non-null  int64
 1   Year       1009674 non-null  int64
 2   isLondon   1009674 non-null  int64
 3   TypeD      1009674 non-null  uint8
 4   TypeF      1009674 non-null  uint8
 5   TypeO      1009674 non-null  uint8
 6   TypeS      1009674 non-null  uint8
 7   TypeT      1009674 non-null  uint8
 8   DurationF  1009674 non-null  uint8
 9   DurationL  1009674 non-null  uint8
 10  DurationU  1009674 non-null  uint8
dtypes: int64(3), uint8(8)
memory usage: 38.5 MB
```

Delete variables we will not use

[16]:

```
del trainData['Year']
del testData['Year']
```

Make a vector for the variables for property price which is the variable we want to predict

[27]:

```
trainPrice = trainData['Price']
testPrice = testData['Price']
```

Fit the model, the model of choice is the random forest model.

[19]:

```
from sklearn.ensemble import RandomForestRegressor
ranForReg = RandomForestRegressor(n_estimators=10,n_jobs =-1)
ranForReg.fit(trainData, trainData['Price'])
```

[19]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=-1, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Find the mean squared errors of the predictions compared to the actual observations.

[21]:

```
from sklearn.metrics import mean_squared_error
myPredictions = ranForReg.predict(trainData)
ranForMSE = mean_squared_error(trainPrice, myPredictions)
ranForRMSE = np.sqrt(ranForMSE)
```

[21]:

```
0.04071408274912438
```

[23]:

```
ranForRMSE.round()
```

[23]:

```
7833.0
```

[31]:

```
ranForRMSE/ trainPrice.mean()
```

[31]:

```
0.04071408274912438
```

[ ]:

```
#Lets see how well the algorithm predicts by using 10-fold cross-validation
import numpy as np
from sklearn.model_selection import cross_val_score
myScores = cross_val_score(ranForReg, trainData, trainPrice, ␣
 ↪scoring="neg_mean_squared_error", cv=10,n_jobs =-1)
ranForRmse = np.sqrt(-myScores)
```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[28]:

```python
#Get root squared mean errors for the test set, that is see how well the model predicts on the testing set and compare to the mean of the price
finalPreds = ranForReg.predict(testData)
finalMSE = mean_squared_error(testPrice, finalPreds)
finalRMSE = np.sqrt(finalMSE)
finalRMSE.round()
```

[28]:

```
7891.0
```

[29]:

```python
finalRMSE/testPrice.mean()
```

[29]:

```
0.02655437610770971
```

[32]:

```python
#Get root mean squared forecasting error
SE = (finalPreds - testPrice) ** 2
SFE = SE.divide(testPrice**2)
MSFE = SFE.mean()
RMSFE = np.sqrt(MSFE)
RMSFE
```

[32]:

```
4.211414618902267e-05
```

[ ]:

```

```