A.Using either the housing dataset

OUTPUT:

In [7]:
```
# take a quick look at the data and it's stats.
housing= load_housing_data()
housing.head()
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

In [8]:
```
# to get quick description of data.
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
```

In [8]:
```
# to get quick description of data.
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [9]:
```
# number of categories that exists in ocean_proximity
housing['ocean_proximity'].value_counts()
```

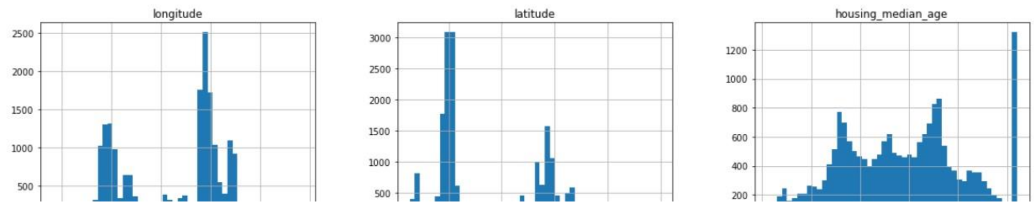
Out[9]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

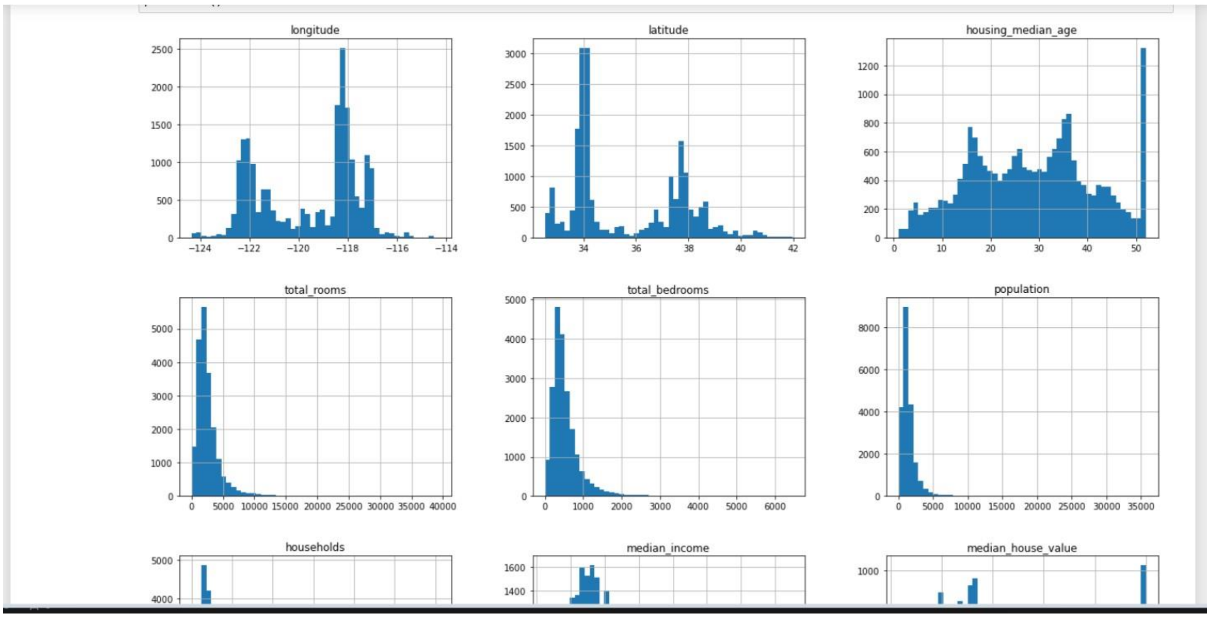
In [10]:
```
# summary of numerical attributes.
housing.describe()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

In [11]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

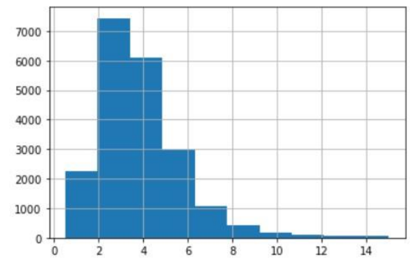




Out[13]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN

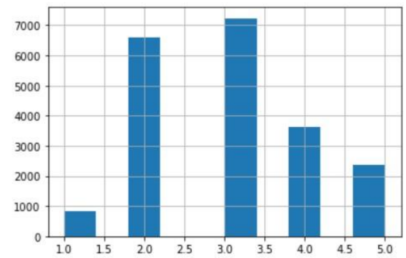
In [15]:
```
housing['median_income'].hist()
plt.show()
```



In [17]:
```
housing["income_cat"].value_counts()
```

Out[17]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

In [18]:
```
housing['income_cat'].hist()
plt.show()
```

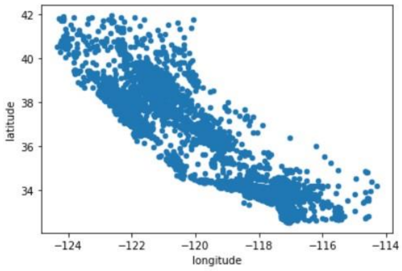


```
Out[20]: 3    0.350533
         2    0.318798
         4    0.176357
         5    0.114341
         1    0.039971
         Name: income_cat, dtype: float64
```

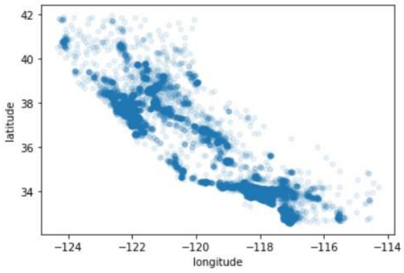
```
In [21]: # Now you should remove the income_cat attribute so the data is back to its original state.
         for set_ in (strat_train_set, strat_test_set):
             set_.drop("income_cat", axis=1, inplace=True)
```

```
In [22]: # let's create copy of the dataset to play with it
         housing= strat_train_set.copy()
```

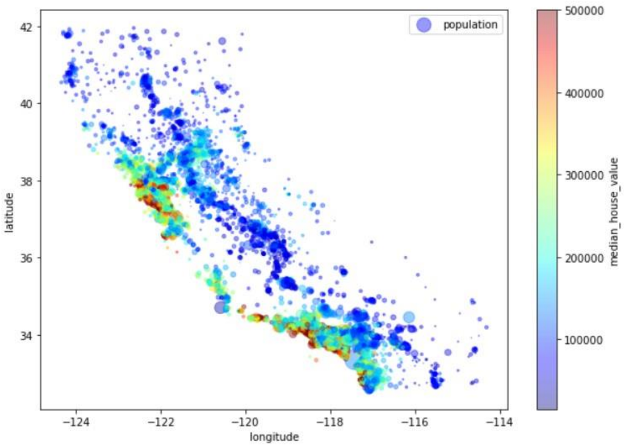
```
In [23]: housing.plot(kind="scatter", x="longitude", y="latitude")
         plt.show()
```




```
In [24]: # it's hard to see any pattern here let's reduce alpha
         housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)
         plt.show()
```



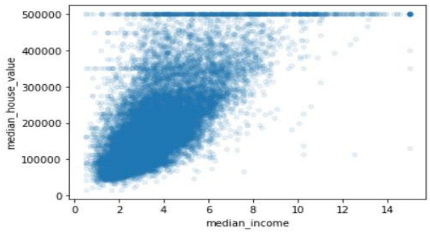

```
In [25]: # let's make it clearer
         housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
                      s=housing["population"]/100, label="population", figsize=(10,7),
                      c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
                      sharex=False)
         plt.legend()
         plt.show()
         # The radius of each circle represents the district’s population (option s), and the color represents the price (option c).
         # We will use a predefined color map (option cmap) called jet, which ranges from blue(low values) to red (high prices).
```




```
In [26]: # let's look for correlations
         corr_matrix= housing.corr()
```

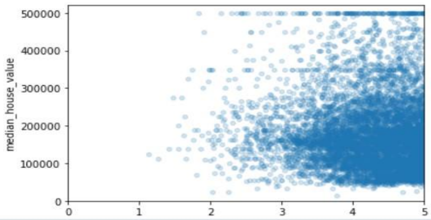
Out[28]:
```
median_house_value    1.000000
median_income         0.687151
total_rooms           0.135140
housing_median_age    0.114146
households            0.064590
total_bedrooms        0.047781
population           -0.026882
longitude            -0.047466
latitude             -0.142673
Name: median_house_value, dtype: float64
```

In [29]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.show()
```



Out[31]:
```
median_house_value          1.000000
median_income               0.687151
rooms_per_household         0.146255
total_rooms                 0.135140
housing_median_age          0.114146
households                  0.064590
total_bedrooms              0.047781
population_per_household   -0.021991
population                 -0.026882
longitude                  -0.047466
latitude                   -0.142673
bedrooms_per_room          -0.259952
Name: median_house_value, dtype: float64
```

In [32]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [33]:
```
housing.describe()
```
Out[33]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	roor
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.000000	16512.000000	
mean	-119.575635	35.639314	28.653404	2622.539789	534.914639	1419.687379	497.011810	3.875884	207005.322372	
std	2.001828	2.137963	12.574819	2138.417080	412.665649	1115.663036	375.696156	1.904931	115701.297250	
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.499900	14999.000000	
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.566950	119800.000000	
50%	-118.510000	34.260000	29.000000	2119.000000	433.000000	1164.000000	408.000000	3.541550	179500.000000	
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.000000	602.000000	4.745325	263900.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.000100	500001.000000	

In [34]:
```
housing = strat_train_set.drop("median_house_value", axis=1) # drop labels for training set
housing_labels = strat_train_set["median_house_value"].copy()
```
In [35]:
```
# DATA Cleaning
# we will fill the the numerical missing values with their medians.
# Scikit-Learn provides a handy class to take care of missing values: SimpleImputer
from sklearn.impute import SimpleImputer
imputer= SimpleImputer(strategy='median')
```
In [39]:
```
imputer.statistics_
```
Out[39]:
```
array([-118.51   ,   34.26   ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```
In [40]:
```
#checking if it is same as the median
housing_num.median().values
```
Out[40]:
```
array([-118.51   ,   34.26   ,   29.     , 2119.     ,  433.     ,
        1164.     ,  408.     ,    3.54155])
```
In [41]:
```
X= imputer.transform(housing_num)
```
In [42]:
```
# HANDLING CATEGORICAL ATTRIBUTES
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```
Out[42]:

	ocean_proximity
12655	INLAND
15502	NEAR OCEAN
2908	INLAND
14053	NEAR OCEAN
20496	<1H OCEAN
1481	NEAR BAY
18125	<1H OCEAN
5830	<1H OCEAN
17989	<1H OCEAN
4861	<1H OCEAN

Out[43]:
```
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

In [44]:
```
#CUSTOM TRANSFORMATIONS
from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

In [47]:
```
housing_prepared
```

Out[47]:
```
array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
         0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
         0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
         0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
         0.        ,  0.        ]])
```

In [48]:
```
# Let's train a linear regression model
from sklearn.linear_model import LinearRegression
lin_reg= LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Out[48]:
```
LinearRegression()
```

In [49]:
```python
#Let's measure RSME(root mean squared error) of our model

from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[49]: 68627.87390018745

In [50]:
```python
from sklearn.tree import DecisionTreeRegressor
tree_reg= DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

Out[50]: DecisionTreeRegressor()

In [51]:
```python
#Let's evatuale on training set
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[51]: 0.0

In [52]:
```python
# cross validation use the train_test_split function to split the training set into a
# smaller training set and a validation set, then train your models against the smaller training
#set and evaluate them against the validation set.
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
```

In [66]:
```python
# let's see the scores
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [71950.33336763 70846.90656375 67743.15019504 70397.56313608
 70125.24497351 77752.48231597 71302.6193184  73142.52222559
 67090.60583868 70416.37529487]
Mean: 71076.78032295093
Standard deviation: 2801.870696246111
```

In [54]:
```python
# let's look for scores for linear regression:
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
# the Decision Tree model is overfitting so badly that it performs worse than the Linear Regression model.
```

```
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.328209818065
```

In [55]:
```python
# let's try Random Forest Regressor
# (Random Forests work by training many Decision Trees on random subsets of the features,
# then averaging out their predictions)
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```
Out[55]:
```
RandomForestRegressor(random_state=42)
```
In [56]:
```python
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```
Out[56]:
```
18650.698705770003
```
In [64]:
```python
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```
```
Scores: [51559.63379638 48737.57100062 47210.51269766 51875.21247297
 47577.50470123 51863.27467888 52746.34645573 50065.1762751
 48664.66818196 54055.90894609]
Mean: 50435.58092066179
Standard deviation: 2203.3381412764606
```

In [55]:
```python
# let's try Random Forest Regressor
# (Random Forests work by training many Decision Trees on random subsets of the features,
# then averaging out their predictions)
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```
Out[55]:
```
RandomForestRegressor(random_state=42)
```
In [56]:
```python
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```
Out[56]:
```
18650.698705770003
```
In [64]:
```python
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```
```
Scores: [51559.63379638 48737.57100062 47210.51269766 51875.21247297
 47577.50470123 51863.27467888 52746.34645573 50065.1762751
 48664.66818196 54055.90894609]
Mean: 50435.58092066179
Standard deviation: 2203.3381412764606
```

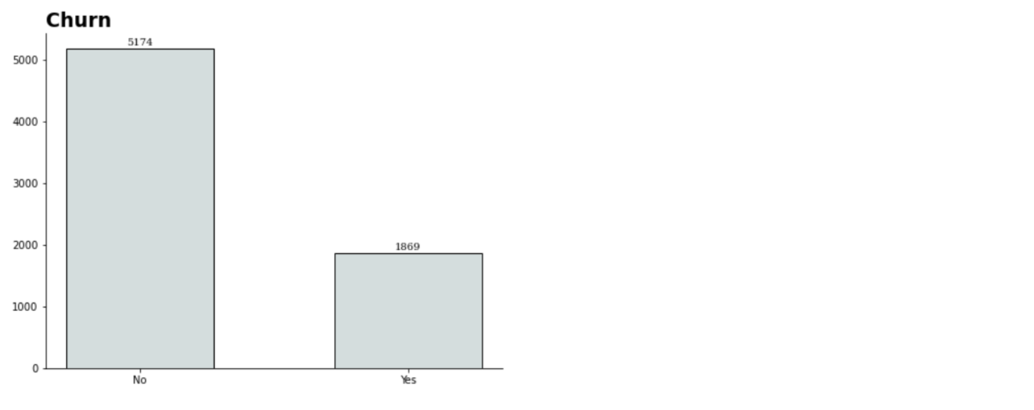
```python
import numpy as np

# Parameter grid for RandomizedSearchCV
param_distribs = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=8),
}

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)

# Best parameters
print("Best parameters:", rnd_search.best_params_)

# Score of each hyperparameter combination tested during randomized search
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print("RMSE:", np.sqrt(-mean_score), "Parameters:", params)
```
```
Best parameters: {'max_features': 7, 'n_estimators': 180}
RMSE: 49117.55344336652 Parameters: {'max_features': 7, 'n_estimators': 180}
RMSE: 51450.63202856348 Parameters: {'max_features': 5, 'n_estimators': 15}
RMSE: 50692.53588182537 Parameters: {'max_features': 3, 'n_estimators': 72}
RMSE: 50783.614493515 Parameters: {'max_features': 5, 'n_estimators': 21}
RMSE: 49162.89877456354 Parameters: {'max_features': 7, 'n_estimators': 122}
RMSE: 50655.798471042704 Parameters: {'max_features': 3, 'n_estimators': 75}
RMSE: 50513.856319990606 Parameters: {'max_features': 3, 'n_estimators': 88}
RMSE: 49521.17201976928 Parameters: {'max_features': 5, 'n_estimators': 100}
RMSE: 50302.90440763418 Parameters: {'max_features': 3, 'n_estimators': 150}
RMSE: 65167.02018649492 Parameters: {'max_features': 5, 'n_estimators': 2}
```

B. Using the customer churn dataset:

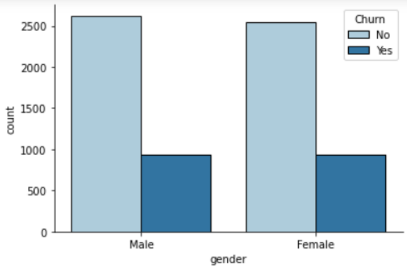
OUTPUT:

```
In [4]: df = pd.read_csv("C:/Users/SREENIJA/Downloads/WA_Fn-UseC_-Telco-Customer-Churn.csv")

In [5]: # Inspect Data
        df.head(2)
        df["SeniorCitizen"]= df["SeniorCitizen"].map({0: "No", 1: "Yes"})

In [6]: # Each row represents a customer, each column contains customer’s attributes described on the column Metadata.
        df.isnull().sum(axis = 0)

Out[6]: customerID          0
        gender              0
        SeniorCitizen       0
        Partner             0
        Dependents          0
        tenure              0
        PhoneService        0
        MultipleLines       0
        InternetService     0
        OnlineSecurity      0
        OnlineBackup        0
        DeviceProtection    0
        TechSupport         0
        StreamingTV         0
        StreamingMovies     0
        Contract            0
        PaperlessBilling    0
        PaymentMethod       0
        MonthlyCharges      0
        TotalCharges        0
        Churn               0
        dtype: int64
```

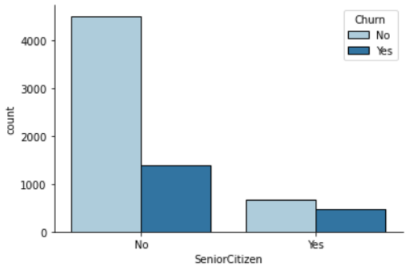
Visualizing our dataset



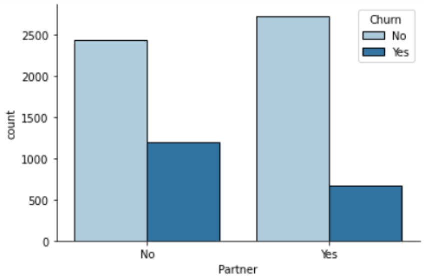
Demographic Variables



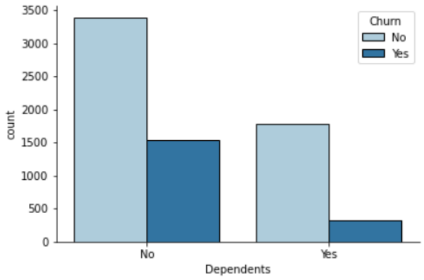
	Churn
('Female', 'No')	0.7308
('Female', 'Yes')	0.2692
('Male', 'No')	0.7384
('Male', 'Yes')	0.2616



	Churn
('No', 'No')	0.7639
('No', 'Yes')	0.2361
('Yes', 'No')	0.5832
('Yes', 'Yes')	0.4168

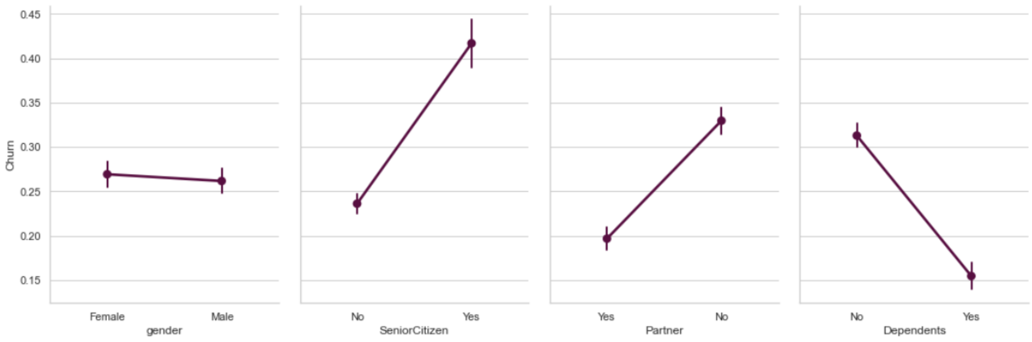


	Churn
('No', 'No')	0.6704
('No', 'Yes')	0.3296
('Yes', 'No')	0.8034
('Yes', 'Yes')	0.1966



	Churn
('No', 'No')	0.6872
('No', 'Yes')	0.3128
('Yes', 'No')	0.8455
('Yes', 'Yes')	0.1545

Out[11]: <seaborn.axisgrid.PairGrid at 0x2a06eb201f0>



In [12]:
```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import math

Male_Churn = df2[df2["gender"] == 'Male'].Churn
Female_Churn = df2[df2["gender"] == 'Female'].Churn

t_statstics = stats.ttest_ind(a= Male_Churn,
                b= Female_Churn,
                equal_var=False)    # Assume samples have equal variance?
t_statstics
```

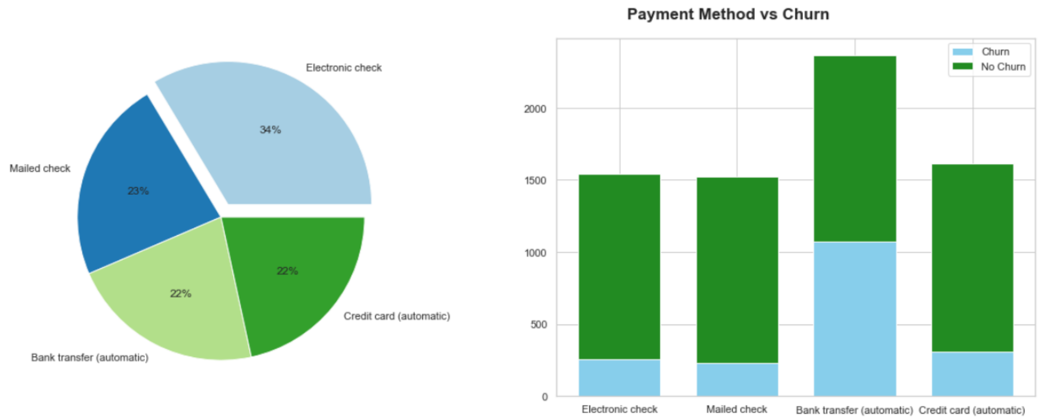
Out[12]: Ttest_indResult(statistic=-0.7226104987857616, pvalue=0.4699432354173566)

In [13]:
```
Dependents_No = df2[df2["Dependents"] == 'No'].Churn
Dependents_Yes = df2[df2["Dependents"] == 'Yes'].Churn

t_statstics1 = stats.ttest_ind(a= Dependents_No, b= Dependents_Yes, equal_var=False)
print(t_statstics1)
t_statstics2 = stats.ttest_ind(a= df2[df2["Partner"] == 'No'].Churn, b=  df2[df2["Partner"] == 'Yes'].Churn, equal_var=False)
print(t_statstics2)
t_statstics3 = stats.ttest_ind(a= df2[df2["SeniorCitizen"] == 'No'].Churn, b=  df2[df2["SeniorCitizen"] == 'Yes'].Churn, equal_va
print(t_statstics3)
```

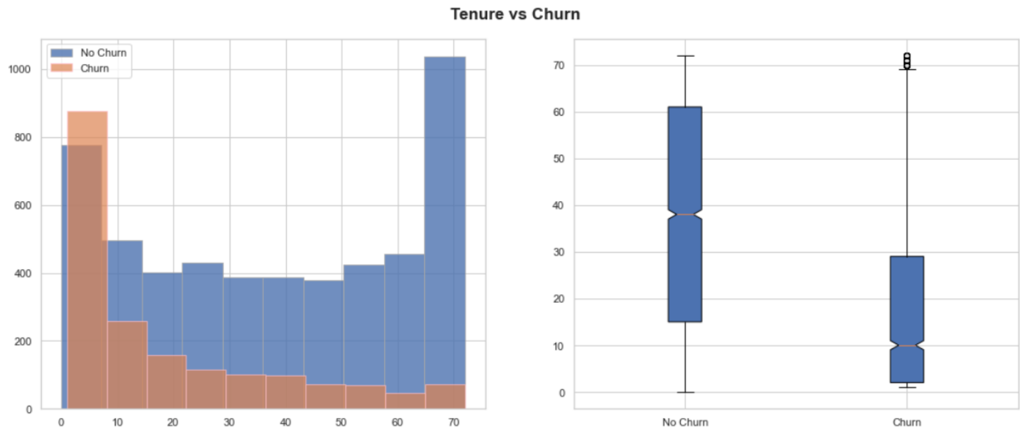
```
Ttest_indResult(statistic=15.409078802902004, pvalue=2.1775286391572522e-52)
Ttest_indResult(statistic=12.841725043203832, pvalue=2.529114349220257e-37)
Ttest_indResult(statistic=-11.58073209133662, pvalue=9.364391561685353e-30)
```

#Customer account information Visualization

Out[14]: Text(0.6, 0.92, 'Payment Method vs Churn')

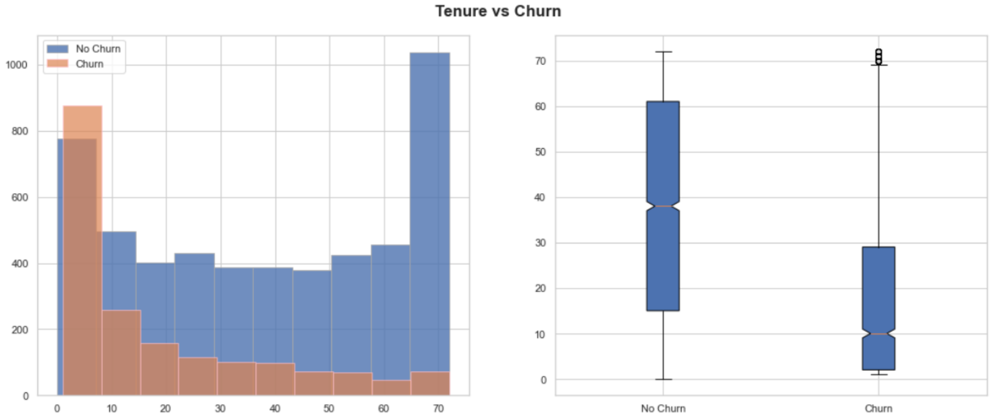


Out[15]: Text(0.45, 0.92, 'Tenure vs Churn')

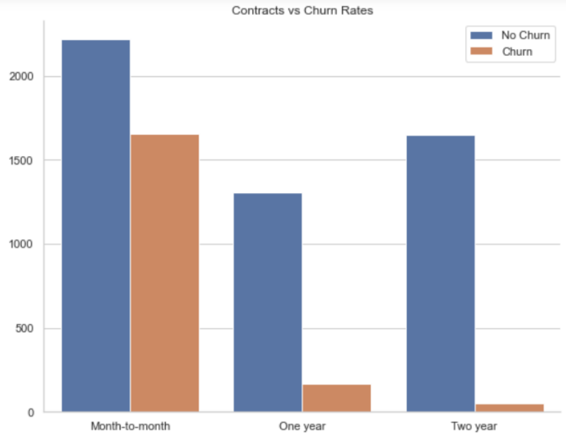


In [16]:
```
plt.figure(figsize=(9,7))
ax = sn.countplot(x="Contract", hue="Churn", data=df).set(title='Contracts vs Churn Rates', xlabel=None, ylabel = None)
sn.despine()
plt.legend(title='', loc='upper right', labels=['No Churn', 'Churn'])
plt.show(g)
```

Out[15]: Text(0.45, 0.92, 'Tenure vs Churn')



In [16]:
```
plt.figure(figsize=(9,7))
ax = sn.countplot(x="Contract", hue="Churn", data=df).set(title='Contracts vs Churn Rates', xlabel=None, ylabel = None)
sn.despine()
plt.legend(title='', loc='upper right', labels=['No Churn', 'Churn'])
plt.show(g)
```

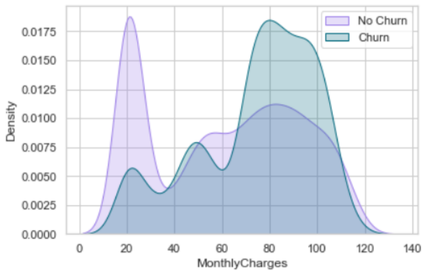


#monthly charges

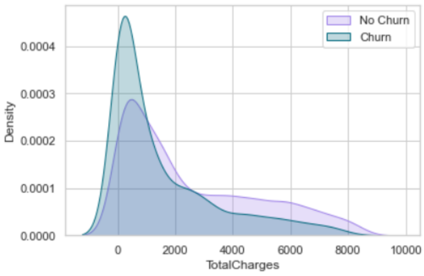
#monthly charges

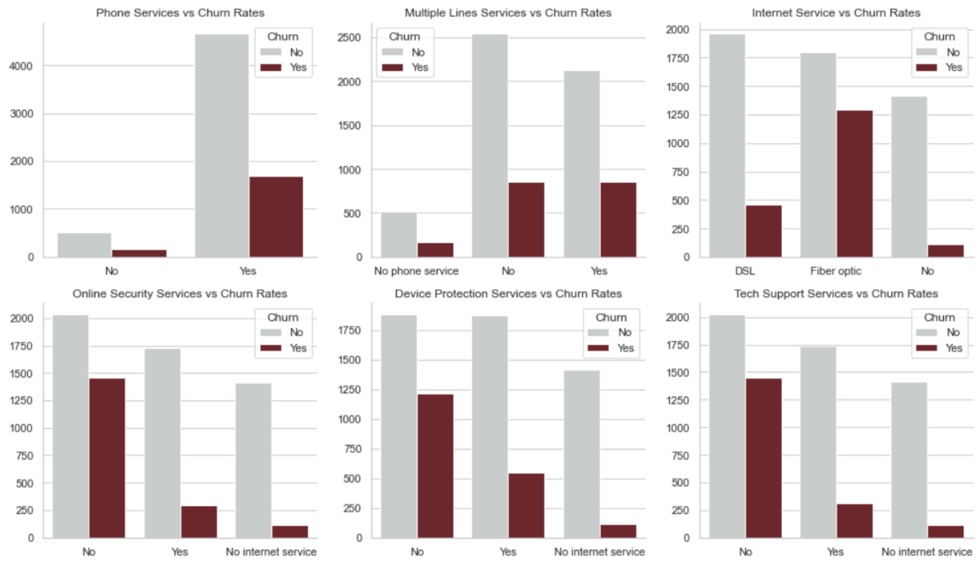
In [17]:
```
ax = sn.kdeplot(Churn_0.MonthlyCharges, color="#9C7FE8", shade = True)
ax = sn.kdeplot(Churn_1.MonthlyCharges, color="#00677C", shade = True)
ax.legend(["No Churn","Churn"],loc='upper right')
```

Out[17]: `<matplotlib.legend.Legend at 0x2a070951ca0>`

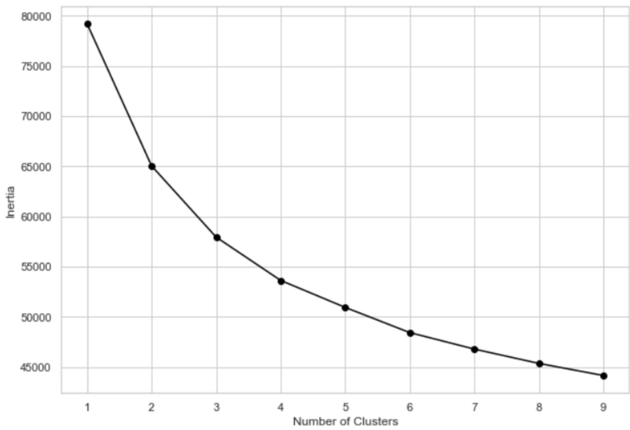


Out[18]: `<matplotlib.legend.Legend at 0x2a0709e7a00>`

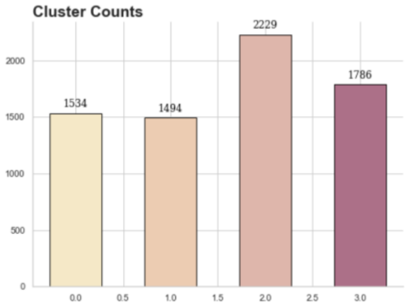


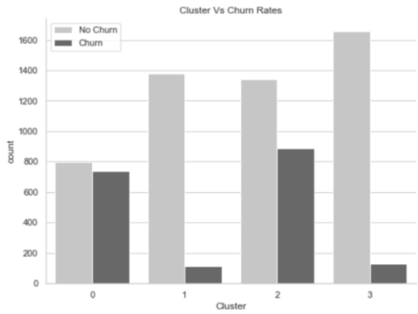


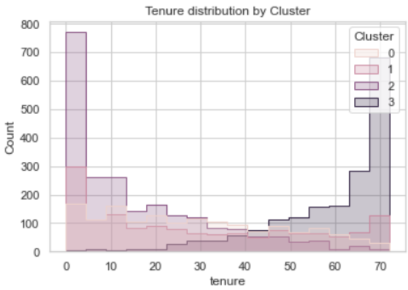
In [22]:
```
optimise_k_means(df_cluster, 10)
```



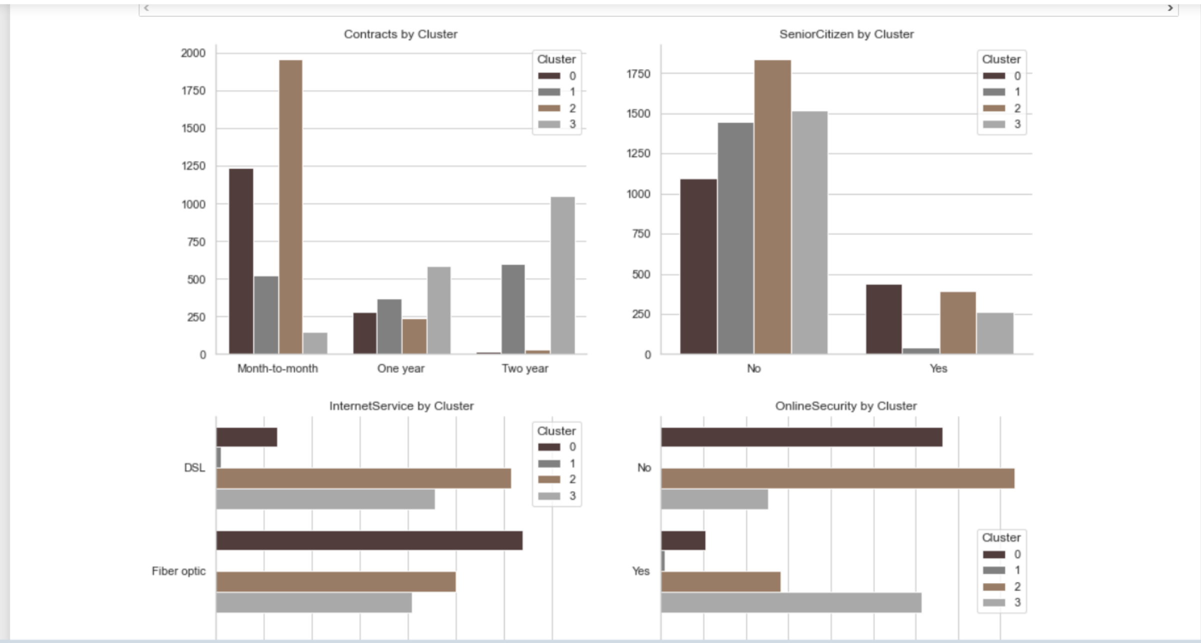
In [23]:
```
# K-Means cluster analysis
kmeans = KMeans(n_clusters = 4,  random_state=10)
kmeans.fit(df_cluster)
# Save cluster group as a column value in our data_frame
df_cluster['Cluster'] = kmeans.labels_
```



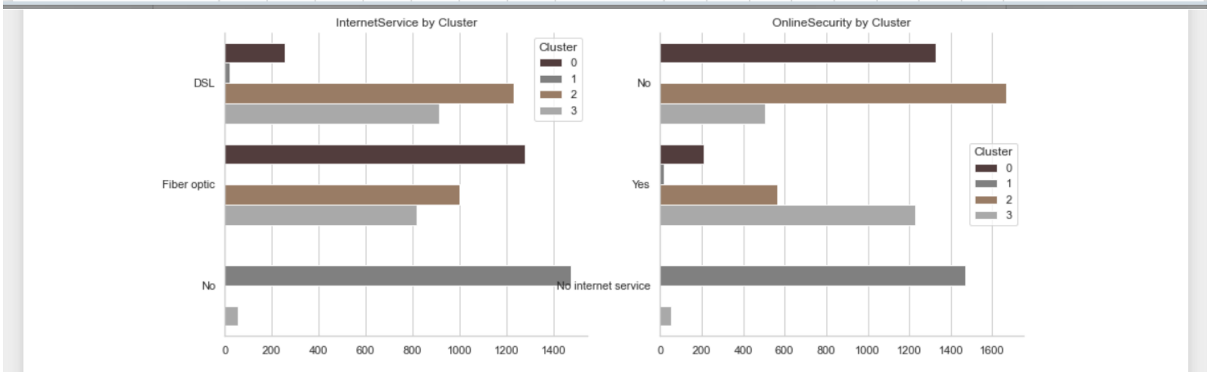




In [26]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14,12))
sn.despine()

# Gray for No Churn, highlight Churn!
colors = ["#553939", "#808080", "#A27B5C","#A9A9A9"]
# Set custom color palette
sn.set_palette(sn.color_palette(colors))
ax = sn.countplot(x="Contract", hue="Cluster", data=df, ax = axes[0,0]).set(title='Contracts by Cluster', xlabel=None, ylabel = N
ax = sn.countplot(x="SeniorCitizen", hue="Cluster", data=df, ax = axes[0,1]).set(title='SeniorCitizen by Cluster', xlabel=None, y
ax = sn.countplot(y='InternetService', hue="Cluster", data=df,ax = axes[1,0]).set(title='InternetService by Cluster', xlabel=None
ax = sn.countplot(y='OnlineSecurity', hue="Cluster", data=df, ax = axes[1,1]).set(title='OnlineSecurity by Cluster', xlabel=None,
sn.despine()
```



Contracts by Cluster
2000
1750
1500
1250
1000
750
500
250
0
Month-to-month
One year
Two year
Cluster
0
1
2
3
SeniorCitizen by Cluster
1750
1500
1250
1000
750
500
250
0
No
Yes
Cluster
0
1
2
3
InternetService by Cluster
DSL
Fiber optic
Cluster
0
1
2
3
OnlineSecurity by Cluster
No
Yes
Cluster
0
1
2
3



InternetService by Cluster
DSL
Fiber optic
No
0
200
400
600
800
1000
1200
1400
Cluster
0
1
2
3
OnlineSecurity by Cluster
No
Yes
No internet service
0
200
400
600
800
1000
1200
1400
1600
Cluster
0
1
2
3