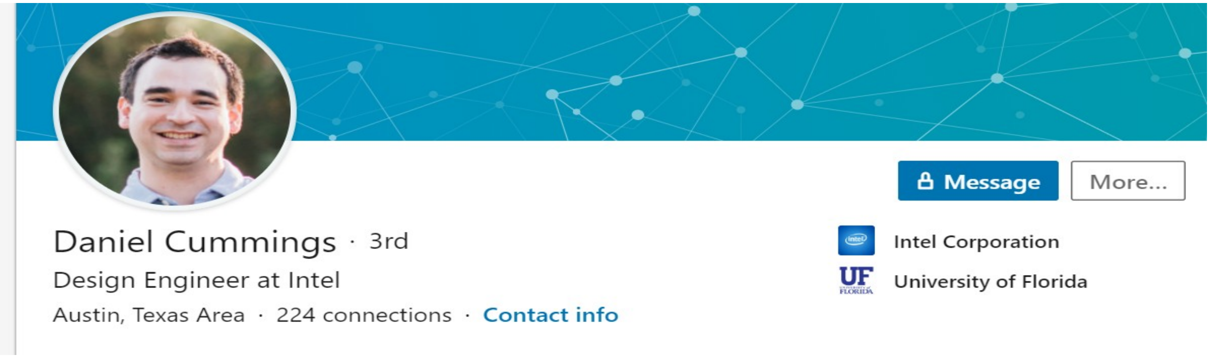
Length Of Stay Prediction

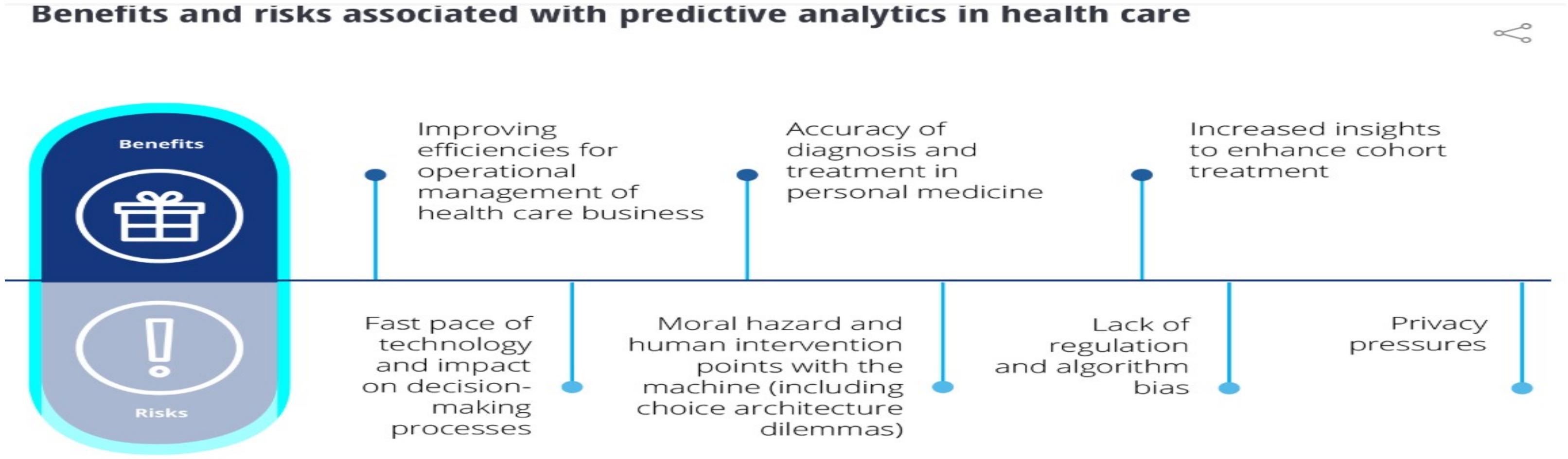
Predicting hospital length-of-stay at time of admission

- https://towardsdatascience.com/predicting-hospital-length-of-stay-at-time-of-admission-55dfdfe69598
- https://github.com/daniel-codes/hospital-los-predictor
- https://github.com/daniel-codes/hospital-los-predictor/blob/master/hospital_los_prediction.ipynb



Predictive Analytics for Healthcare

- Predictive analytics is an increasingly important tool in the healthcare field since modern machine learning (ML) methods can use large amounts of available data to predict individual outcomes for patients.



https://www2.deloitte.com/us/en/insights/topics/analytics/predictive-analytics-health-care-value-risks.html

LOS

- hospital **length-of-stay** (LOS)
  - LOS is defined as the time between hospital admission and discharge measured in days.
- U.S. hospital stays cost the health system at least $377.5 billion per year and recent Medicare legislation standardizes payments for procedures performed, regardless of the number of days a patient spends in the hospital. This incentivizes hospitals to identify patients of high LOS risk at the time of admission. Once identified, patients with high LOS risk can have their treatment plan optimized to minimize LOS and lower the chance of getting a hospital-acquired condition such as staph infection. Another benefit is that prior knowledge of LOS can aid in logistics such as room and bed allocation planning.

https://www.healthcatalyst.com/success_stories/reducing-length-of-stay-in-hospital

Goal

- The goal is to create a model that predicts the length of stay for each patient at time of admission

Import libraries or packages

```python
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import GridSearchCV
```

MIMIC III data

- We need the following tables for this tutorial
  - Unzip the files and upload them to google colab

ADMISSIONS.csv

DIAGNOSES_ICD.csv

ICUSTAYS.csv

PATIENTS.csv

Load MIMIC tables

```
from google.colab import files
uploaded = files.upload()
```

Choose Files ADMISSIONS.csv

- **ADMISSIONS.csv**(application/vnd.ms-excel) - 12548562 bytes, last modified: 9/2/2016 - 16% done

```
Saving ADMISSIONS.csv to ADMISSIONS.csv
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files DIAGNOSES_ICD.csv

- **DIAGNOSES_ICD.csv**(application/vnd.ms-excel) - 19137527 bytes, last modified: 9/2/2016 - 8% done

```
Saving DIAGNOSES_ICD.csv to DIAGNOSES_ICD.csv
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files ICUSTAYS.csv

- **ICUSTAYS.csv**(application/vnd.ms-excel) - 6357077 bytes, last modified: 9/2/2016 - 26% done

```
Saving ICUSTAYS.csv to ICUSTAYS.csv
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files PATIENTS.csv

- **PATIENTS.csv**(application/vnd.ms-excel) - 2628900 bytes, last modified: 9/2/2016 - 100% done

```
Saving PATIENTS.csv to PATIENTS.csv
```

Read data into google colab

```
# Primary Admissions information
df = pd.read_csv('ADMISSIONS.csv')

# Patient specific info such as gender
df_pat = pd.read_csv('PATIENTS.csv')

# Diagnosis for each admission to hospital
df_diagcode = pd.read_csv('DIAGNOSES_ICD.csv')

# Intensive Care Unit (ICU) for each admission to hospital
df_icu = pd.read_csv('ICUSTAYS.csv')
```

Data Exploration and feature engineering

- From MIMIC: The ADMISSIONS table gives information regarding a patient's admission to the hospital. Since each unique hospital visit for a patient is assigned a unique HADM_ID, the ADMISSIONS table can be considered as a definition table for HADM_ID. Information available includes timing information for admission and discharge, demographic information, the source of the admission, and so on.

ADMISSIONS.csv Exploration

```
print('Dataset has {} number of unique admission events.'.format(df['HADM_ID'].nunique()))
print('Dataset has {} number of unique patients.'.format(df['SUBJECT_ID'].nunique()))
```

```
Dataset has 58976 number of unique admission events.
Dataset has 46520 number of unique patients.
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58976 entries, 0 to 58975
Data columns (total 19 columns):
ROW_ID                  58976 non-null int64
SUBJECT_ID              58976 non-null int64
HADM_ID                 58976 non-null int64
ADMITTIME               58976 non-null object
DISCHTIME               58976 non-null object
DEATHTIME               5854 non-null object
ADMISSION_TYPE          58976 non-null object
ADMISSION_LOCATION      58976 non-null object
DISCHARGE_LOCATION      58976 non-null object
INSURANCE               58976 non-null object
LANGUAGE                33644 non-null object
RELIGION                58518 non-null object
MARITAL_STATUS          48848 non-null object
ETHNICITY               58976 non-null object
EDREGTIME               30877 non-null object
EDOUTTIME               30877 non-null object
DIAGNOSIS               58951 non-null object
HOSPITAL_EXPIRE_FLAG    58976 non-null int64
HAS_CHARTEVENTS_DATA    58976 non-null int64
dtypes: int64(5), object(14)
memory usage: 8.5+ MB
```

Feature Engineering

- The first task is to figure out a way to calculate the LOS. LOS is defined as the time between admission and discharge from the hospital.

```
df.head()
```

	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	DEATHTIME	ADMISSION_TYPE	ADMISSION_LOCATION
0	21	22	165315	2196-04-09 12:26:00	2196-04-10 15:54:00	NaN	EMERGENCY	EMERGENCY ROOM ADMIT
1	22	23	152223	2153-09-03 07:15:00	2153-09-08 19:10:00	NaN	ELECTIVE	PHYS REFERRAL/NORMAL DELI
2	23	23	124321	2157-10-18 19:34:00	2157-10-25 14:00:00	NaN	EMERGENCY	TRANSFER FROM HOSP/EXTRAM
3	24	24	161859	2139-06-06 16:14:00	2139-06-09 12:48:00	NaN	EMERGENCY	TRANSFER FROM HOSP/EXTRAM

```python
# Convert admission and discharge times to datatime type
df['ADMITTIME'] = pd.to_datetime(df['ADMITTIME'])
df['DISCHTIME'] = pd.to_datetime(df['DISCHTIME'])

# Convert timedelta type into float 'days', 86400 seconds in a day
df['LOS'] = (df['DISCHTIME'] - df['ADMITTIME']).dt.total_seconds()/86400
```

```python
# Verify
df[['ADMITTIME', 'DISCHTIME', 'LOS']].head()
```

	ADMITTIME	DISCHTIME	LOS
0	2196-04-09 12:26:00	2196-04-10 15:54:00	1.144444
1	2153-09-03 07:15:00	2153-09-08 19:10:00	5.496528
2	2157-10-18 19:34:00	2157-10-25 14:00:00	6.768056
3	2139-06-06 16:14:00	2139-06-09 12:48:00	2.856944
4	2160-11-02 02:06:00	2160-11-05 14:55:00	3.534028

```python
df['LOS'].describe()
```

```
count    58976.000000
mean        10.133916
std         12.456682
min         -0.945139
25%          3.743750
50%          6.467014
75%         11.795139
max        294.660417
Name: LOS, dtype: float64
```

```python
# Look at what is happening with negative LOS values
df[df['LOS'] < 0]
```

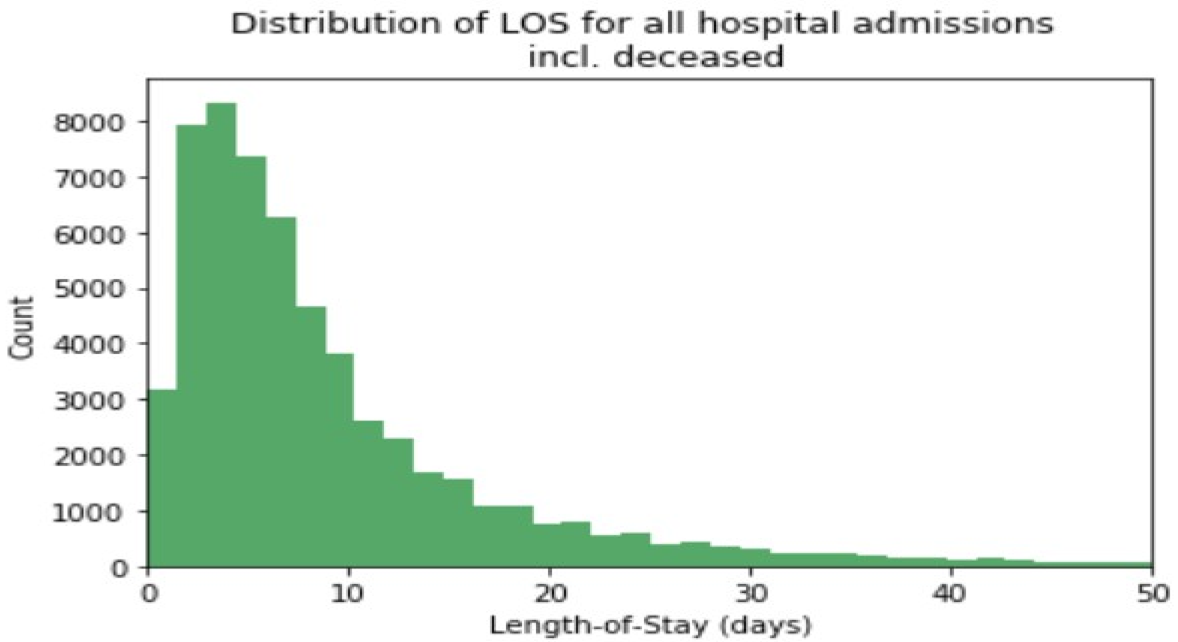
	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	DEATHTIME	ADMISSION_TYPE	ADMISSION_LOCATION
425	534	417	102633	2177-03-23 16:17:00	2177-03-23 07:20:00	2177-03-23 07:20:00	URGENT	PHYS REFERRAL/NORMAL DELI
456	237	181	102631	2153-10-12 09:49:00	2153-10-12 06:29:00	2153-10-12 06:29:00	EMERGENCY	EMERGENCY ROOM ADMIT
692	644	516	187482	2197-07-31 20:18:00	2197-07-31 01:10:00	2197-07-31 01:10:00	EMERGENCY	EMERGENCY ROOM ADMIT

```python
# Drop rows with negative LOS, usually related to a time of death before admission
df['LOS'][df['LOS'] > 0].describe()
```

```
count    58878.000000
mean        10.151266
std         12.459774
min          0.001389
25%          3.755556
50%          6.489583
75%         11.805556
max        294.660417
Name: LOS, dtype: float64
```

```python
# Drop LOS < 0
df = df[df['LOS'] > 0]
```

```python
# Plot LOS Distribution
plt.hist(df['LOS'], bins=200, color = '#55a868')
plt.xlim(0, 50)
plt.title('Distribution of LOS for all hospital admissions \n incl. deceased')
plt.ylabel('Count')
plt.xlabel('Length-of-Stay (days)')
plt.tick_params(top=False, right=False)
plt.show();
```



Deathtime

```
print("{} of {} patients died in the hospital".format(df['DECEASED'].sum(),
                                                      df['SUBJECT_ID'].nunique()))
```

```
5774 of 46445 patients died in the hospital
```

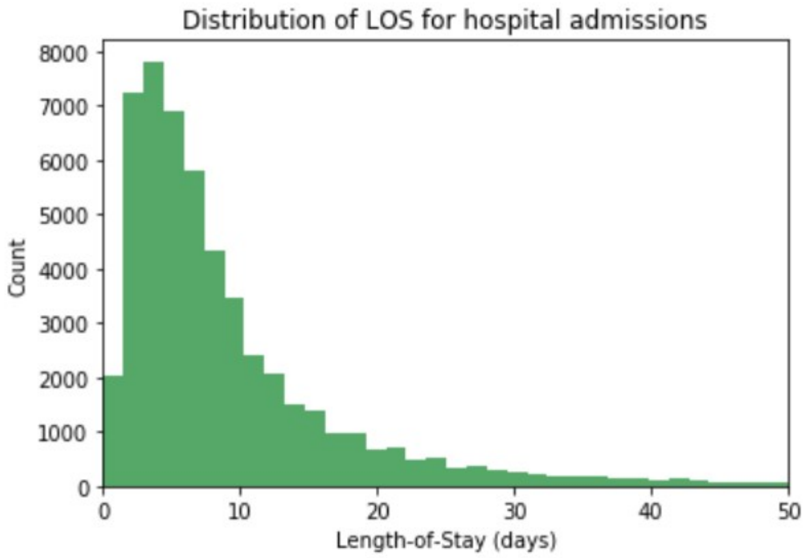
```
# Look at statistics less admissions resulting in death
df['LOS'].loc[df['DECEASED'] == 0].describe()
```

```
count    53104.000000
mean        10.138174
std         12.284461
min          0.014583
25%          3.866667
50%          6.565972
75%         11.711632
max        294.660417
Name: LOS, dtype: float64
```

```python
# Hospital LOS metrics for later comparison
actual_mean_los = df['LOS'].loc[df['DECEASED'] == 0].mean()
actual_median_los = df['LOS'].loc[df['DECEASED'] == 0].median()

print(actual_mean_los)
print(actual_median_los)
```

```
10.138173704219758
6.565972222222222
```

```python
plt.hist(df['LOS'].loc[df['DECEASED'] == 0], bins=200, color = '#55a868')
plt.xlim(0, 50)
plt.title('Distribution of LOS for hospital admissions')
plt.ylabel('Count')
plt.xlabel('Length-of-Stay (days)')
plt.tick_params(top=False, right=False)
plt.show();
```



Ethnicity

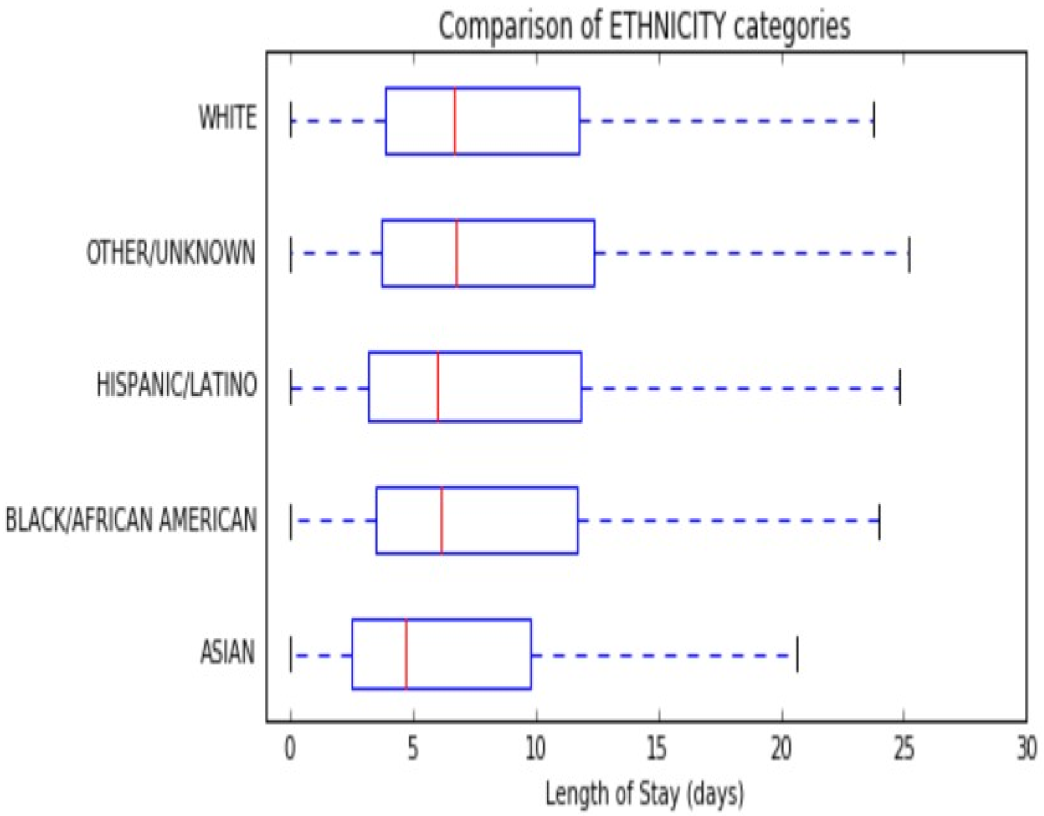
```
df['ETHNICITY'].value_counts()
```

```
WHITE                                40939
BLACK/AFRICAN AMERICAN                5434
UNKNOWN/NOT SPECIFIED                 4502
HISPANIC OR LATINO                    1693
ASIAN                                 1508
OTHER                                 1507
UNABLE TO OBTAIN                       809
PATIENT DECLINED TO ANSWER             559
ASIAN - CHINESE                        277
HISPANIC/LATINO - PUERTO RICAN         232
BLACK/CAPE VERDEAN                     200
WHITE - RUSSIAN                        164
MULTI RACE ETHNICITY                   130
```

```python
# Compress the number of ethnicity categories
df['ETHNICITY'].replace(regex=r'^ASIAN\D*', value='ASIAN', inplace=True)
df['ETHNICITY'].replace(regex=r'^WHITE\D*', value='WHITE', inplace=True)
df['ETHNICITY'].replace(regex=r'^HISPANIC\D*', value='HISPANIC/LATINO', inplace=True)
df['ETHNICITY'].replace(regex=r'^BLACK\D*', value='BLACK/AFRICAN AMERICAN', inplace=True)
df['ETHNICITY'].replace(['UNABLE TO OBTAIN', 'OTHER', 'PATIENT DECLINED TO ANSWER',
                         'UNKNOWN/NOT SPECIFIED'], value='OTHER/UNKNOWN', inplace=True)
df['ETHNICITY'].loc[~df['ETHNICITY'].isin(df['ETHNICITY'].value_counts().nlargest(5).index.tolist())] = 'OTHER/UNKNOWN'
df['ETHNICITY'].value_counts()
```

```
WHITE                     41268
OTHER/UNKNOWN              7700
BLACK/AFRICAN AMERICAN     5779
HISPANIC/LATINO            2125
ASIAN                      2006
Name: ETHNICITY, dtype: int64
```

```python
# Re-usable plotting function
def plot_los_groupby(variable, size=(7,4)):
    '''
    Plot Median LOS by df categorical series name
    '''
    results = df[[variable, 'LOS']].groupby(variable).median().reset_index()
    values = list(results['LOS'].values)
    labels = list(results[variable].values)

    fig, ax = plt.subplots(figsize=size)
    ind = range(len(results))
    ax.barh(ind, values, align='center', height=0.6, color = '#55a868', alpha=0.8)
    ax.set_yticks(ind)
    ax.set_yticklabels(labels)
    ax.set_xlabel('Median Length of Stay (days)')
    ax.tick_params(left=False, top=False, right=False)
    ax.set_title('Comparison of {} labels'.format(variable))

    plt.tight_layout()
    plt.show();
```

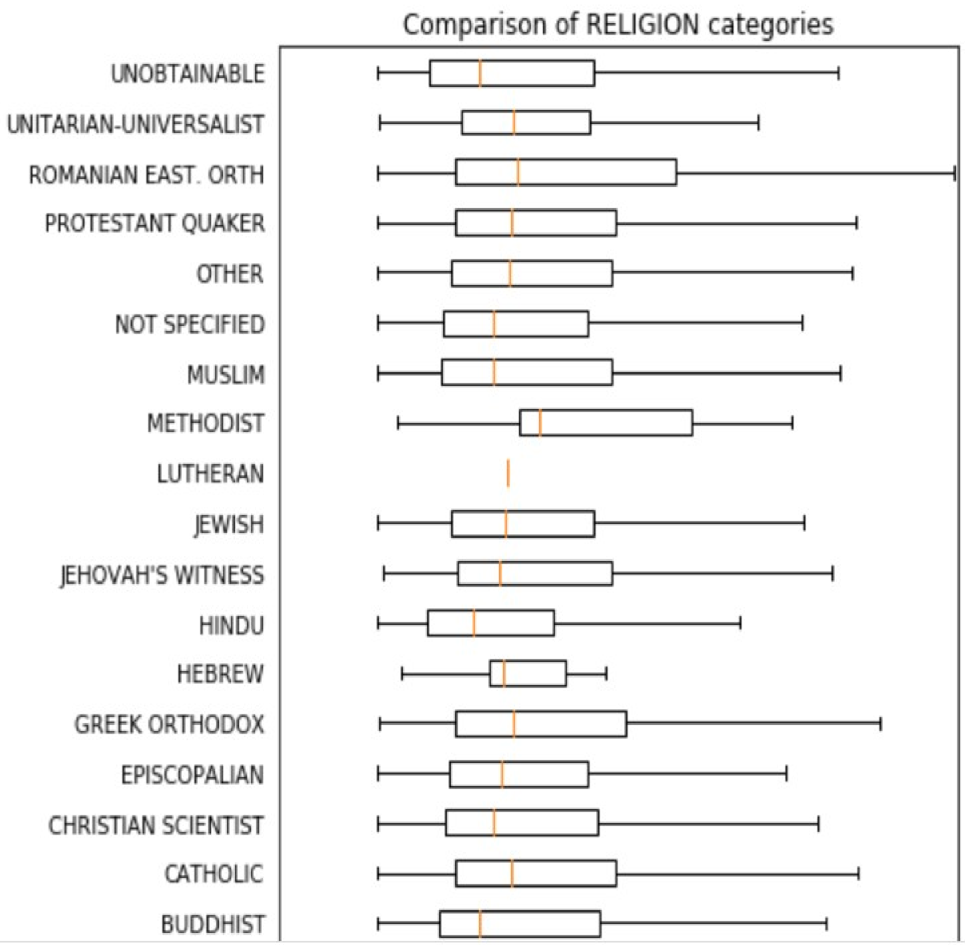


Religion

```
df['RELIGION'].value_counts()
```

```
CATHOLIC                  20580
NOT SPECIFIED             11738
UNOBTAINABLE               8242
PROTESTANT QUAKER          7121
JEWISH                     5307
OTHER                      2695
EPISCOPALIAN                771
GREEK ORTHODOX              459
CHRISTIAN SCIENTIST         429
BUDDHIST                    267
MUSLIM                      225
JEHOVAH'S WITNESS           139
UNITARIAN-UNIVERSALIST      124
HINDU                       113
```

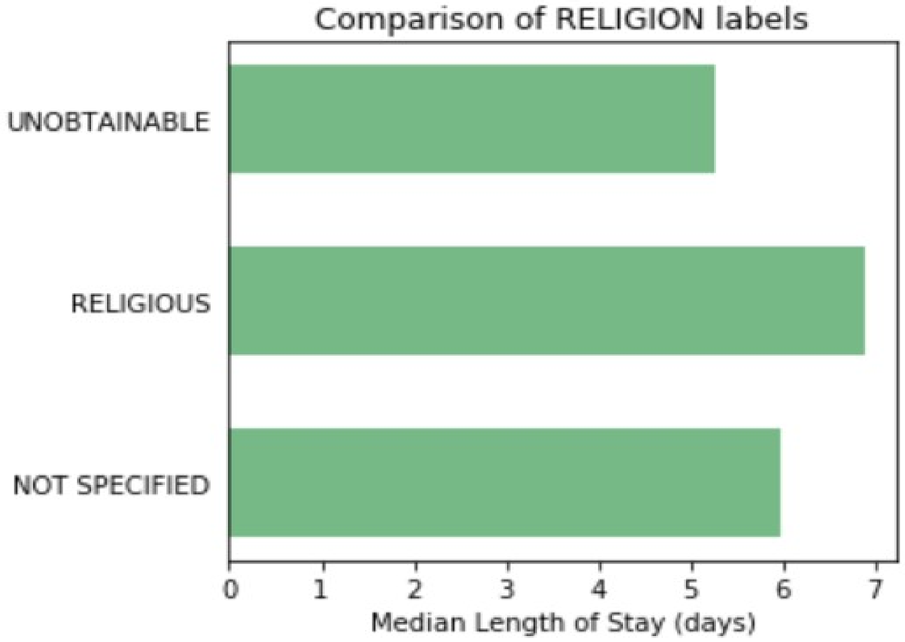
```
boxplot_los_groupby('RELIGION', los_range=(-5, 30), size=(7, 7))
```



```python
# Reduce categories to terms of religious or not
# I tested with and without category reduction, with little change in R2 score
df['RELIGION'].loc[~df['RELIGION'].isin(['NOT SPECIFIED', 'UNOBTAINABLE'])] = 'RELIGIOUS'

print(df['RELIGION'].value_counts())
print(df['RELIGION'].value_counts()[0]/len(df['RELIGION']))
print(df['RELIGION'].value_counts()[1]/len(df['RELIGION']))
print(df['RELIGION'].value_counts()[2]/len(df['RELIGION']))
```

```
RELIGIOUS        38898
NOT SPECIFIED    11738
UNOBTAINABLE      8242
Name: RELIGION, dtype: int64
0.660654234179150l
0.1993613913516084
0.13998437446924147
```

```python
# Look at median LOS for groups
plot_los_groupby('RELIGION', size=(5,4))
```

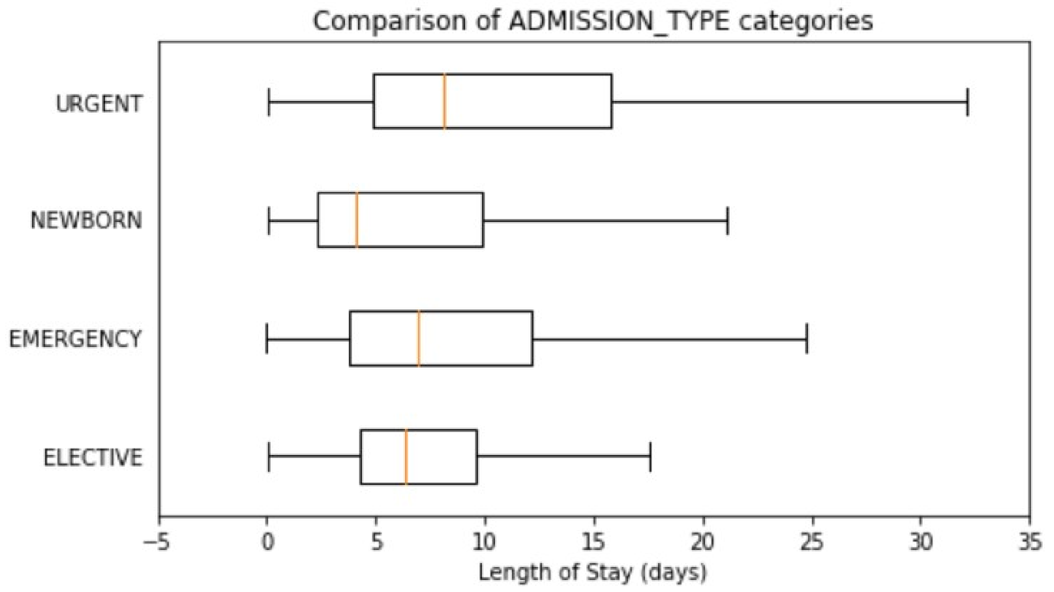


Admission type

```
df['ADMISSION_TYPE'].value_counts()
```

```
EMERGENCY    41989
NEWBORN       7854
ELECTIVE      7702
URGENT        1333
Name: ADMISSION_TYPE, dty
```

```
boxplot_los_groupby('ADMISSION_TYPE', los_range=(-5, 35), size=(7, 4))
```

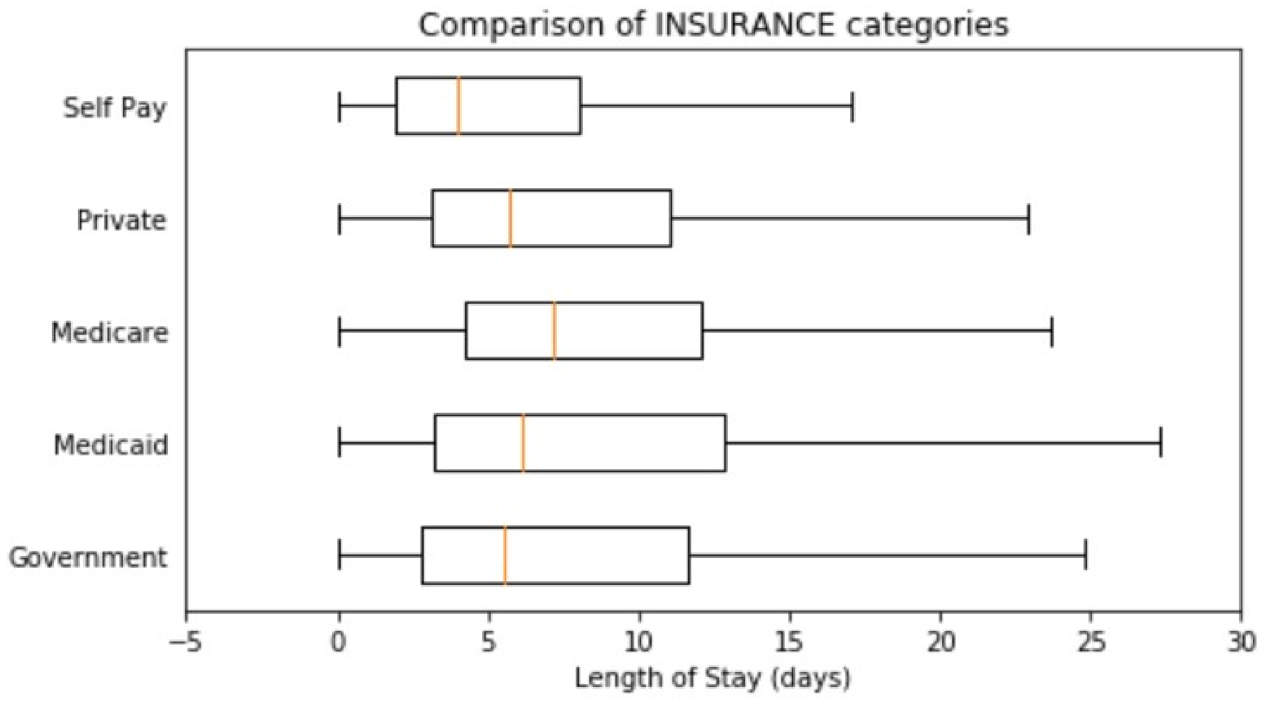


Insurance

```
df['INSURANCE'].value_counts()
```

```
Medicare      28174
Private       22542
Medicaid       5778
Government     1781
Self Pay        603
Name: INSURANCE, dtype: int
```

```
boxplot_los_groupby('INSURANCE', los_range=(-5, 30), size=(7, 4))
```



Marital Status

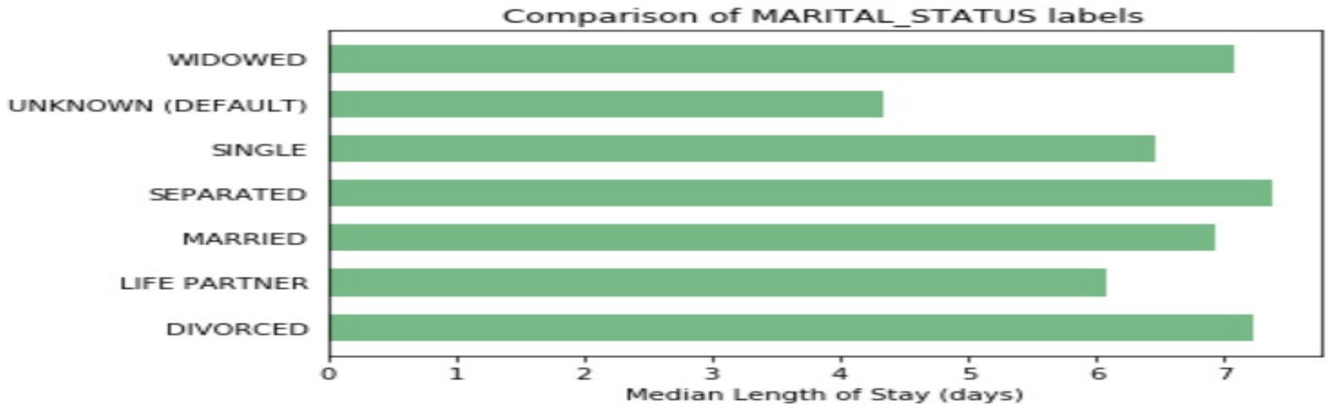
```
df['MARITAL_STATUS'].value_counts(dropna=False)
```

```
MARRIED              24199
SINGLE               13238
NaN                  10097
WIDOWED               7204
DIVORCED              3211
SEPARATED              571
UNKNOWN (DEFAULT)      343
LIFE PARTNER            15
Name: MARITAL_STATUS, dtype: int64
```

```
# Fix NaNs and file under 'UNKNOWN'
df['MARITAL_STATUS'] = df['MARITAL_STATUS'].fillna('UNKNOWN (DEFAULT)')
df['MARITAL_STATUS'].value_counts(dropna=False)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: Setting
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
  """Entry point for launching an IPython kernel.
MARRIED              24199
SINGLE               13238
UNKNOWN (DEFAULT)    10440
WIDOWED               7204
DIVORCED              3211
SEPARATED              571
LIFE PARTNER            15
Name: MARITAL_STATUS, dtype: int64
```

```
plot_los_groupby('MARITAL_STATUS')
```



DIAGNOSES_ICD.csv Exploration

- This section explore the ICUSTAYS.csv table of the MIMIC-III dataset.

```
df_diagcode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 651047 entries, 0 to 651046
Data columns (total 5 columns):
ROW_ID        651047 non-null int64
SUBJECT_ID    651047 non-null int64
HADM_ID       651047 non-null int64
SEQ_NUM       651000 non-null float64
ICD9_CODE     651000 non-null object
dtypes: float64(1), int64(3), object(1)
memory usage: 24.8+ MB
```

```
print('There are {} unique ICD9 codes in this dataset.'.format(df_diagcode['ICD9_CODE'].value_counts().count())
```

```
There are 6984 unique ICD9 codes in this dataset.
```

ICD-9 Code Feature Engineering

Because it's not feasible to have 6984 unique values to use as features for predicting LOS, I need to reduce the diagnosis into more general categories. After researching the ICD9 approach, I discovery that they arranged into super categories as the following (source):

- 001–139: infectious and parasitic diseases
- 140–239: neoplasms
- 240–279: endocrine, nutritional and metabolic diseases, and immunity disorders
- 280–289: diseases of the blood and blood-forming organs
- 290–319: mental disorders
- 320–389: diseases of the nervous system and sense organs
- 390–459: diseases of the circulatory system
- 460–519: diseases of the respiratory system
- 520–579: diseases of the digestive system
- 580–629: diseases of the genitourinary system
- 630–679: complications of pregnancy, childbirth, and the puerperium
- 680–709: diseases of the skin and subcutaneous tissue
- 710–739: diseases of the musculoskeletal system and connective tissue
- 740–759: congenital anomalies
- 760–779: certain conditions originating in the perinatal period
- 780–799: symptoms, signs, and ill-defined conditions
- 800–999: injury and poisoning

E and V codes: external causes of injury and supplemental classification, *using 999 as placeholder even though it overlaps with complications of medical care*

Group ICD9 codes

```
[49] # Filter out E and V codes since processing will be done on the numeric first 3 values
     df_diagcode['recode'] = df_diagcode['ICD9_CODE']
     df_diagcode['recode'] = df_diagcode['recode'][~df_diagcode['recode'].str.contains("[a-zA-Z]").fillna(False)]
     df_diagcode['recode'].fillna(value='999', inplace=True)
```

```
[50] # https://stackoverflow.com/questions/46168450/replace-specific-range-of-values-in-data-frame-pandas
     df_diagcode['recode'] = df_diagcode['recode'].str.slice(start=0, stop=3, step=1)
     df_diagcode['recode'] = df_diagcode['recode'].astype(int)
```

Group ICD9 codes

```python
# ICD-9 Main Category ranges
icd9_ranges = [(1, 140), (140, 240), (240, 280), (280, 290), (290, 320), (320, 390),
               (390, 460), (460, 520), (520, 580), (580, 630), (630, 680), (680, 710),
               (710, 740), (740, 760), (760, 780), (780, 800), (800, 1000), (1000, 2000)]

# Associated category names
diag_dict = {0: 'infectious', 1: 'neoplasms', 2: 'endocrine', 3: 'blood',
             4: 'mental', 5: 'nervous', 6: 'circulatory', 7: 'respiratory',
             8: 'digestive', 9: 'genitourinary', 10: 'pregnancy', 11: 'skin',
             12: 'muscular', 13: 'congenital', 14: 'prenatal', 15: 'misc',
             16: 'injury', 17: 'misc'}

# Re-code in terms of integer
for num, cat_range in enumerate(icd9_ranges):
    df_diagcode['recode'] = np.where(df_diagcode['recode'].between(cat_range[0],cat_range[1]),
            num, df_diagcode['recode'])

# Convert integer to category name using diag_dict
df_diagcode['recode'] = df_diagcode['recode']
df_diagcode['cat'] = df_diagcode['recode'].replace(diag_dict)
```

ICD-9 chapters

Chapter	Block	Title
I	001–139	Infectious and Parasitic Diseases
II	140–239	Neoplasms
III	240–279	Endocrine, Nutritional and Metabolic Diseases, and Immunity Disorders
IV	280–289	Diseases of the Blood and Blood-forming Organs
V	290–319	Mental Disorders
VI	320–389	Diseases of the Nervous System and Sense Organs
VII	390–459	Diseases of the Circulatory System
VIII	460–519	Diseases of the Respiratory System
IX	520–579	Diseases of the Digestive System
X	580–629	Diseases of the Genitourinary System
XI	630–679	Complications of Pregnancy, Childbirth, and the Puerperium
XII	680–709	Diseases of the Skin and Subcutaneous Tissue
XIII	710–739	Diseases of the Musculoskeletal System and Connective Tissue
XIV	740–759	Congenital Anomalies
XV	760–779	Certain Conditions originating in the Perinatal Period
XVI	780–799	Symptoms, Signs and Ill-defined Conditions
XVII	800–999	Injury and Poisoning
	E800–E999	Supplementary Classification of External Causes of Injury and Poisoning
	V01–V82	Supplementary Classification of Factors influencing Health Status and Contact with Health Services
	M8000–M9970	Morphology of Neoplasms

- For each admission, there could be (and usually is) more than one diagnosis. Often, there are more than 1 diagnoses for 1 category. Therefore, I need to create a dummy matrix that highlights all the diagnoses for each admission. This should not be done on the SUBJECT_ID since each patient could have different diagnoses for each admission.

```
# Verify
df_diagcode.head()
```

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	recode	cat
0	1297	109	172335	1.0	40301	6	circulatory
1	1298	109	172335	2.0	486	7	respiratory
2	1299	109	172335	3.0	58281	9	genitourinary
3	1300	109	172335	4.0	5855	9	genitourinary
4	1301	109	172335	5.0	4254	6	circulatory

```
# Create list of diagnoses for each admission
hadm_list = df_diagcode.groupby('HADM_ID')['cat'].apply(list).reset_index()
hadm_list.head()
```

	HADM_ID	cat
0	100001	[endocrine, nervous, genitourinary, digestive,...
1	100003	[digestive, blood, infectious, digestive, circ...
2	100006	[respiratory, respiratory, respiratory, neopla...
3	100007	[digestive, digestive, injury, respiratory, ci...
4	100009	[circulatory, injury, circulatory, endocrine, ...

```python
# Convert diagnoses list into hospital admission-item matrix
hadm_item = pd.get_dummies(hadm_list['cat'].apply(pd.Series).stack()).sum(level=0)
hadm_item.head()
```

	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious	injury
0	0	2	0	2	5	2	0	2
1	1	2	0	4	0	0	1	0
2	0	0	0	0	1	0	0	2
3	0	1	0	2	0	0	0	1
4	1	7	0	0	3	0	0	7

```python
# Join back with HADM_ID, will merge with main admissions DF later
hadm_item = hadm_item.join(hadm_list['HADM_ID'], how="outer")
hadm_item.head()
```

digestive	endocrine	genitourinary	infectious	injury	mental	misc	muscular	neoplasms	nervous	pregnancy	prenatal	respiratory	skin	HADM_ID
2	5	2	0	2	0	0	0	0	2	0	0	0	1	100001
4	0	0	1	0	0	1	0	0	0	0	0	0	0	100003
0	1	0	0	2	1	1	0	1	0	0	0	3	0	100006
2	0	0	0	1	0	0	0	0	0	0	0	1	0	100007
0	3	0	0	7	0	0	0	0	0	0	0	0	0	100009

```
[101] # Merge with main admissions df
      df = df.merge(hadm_item, how='inner', on='HADM_ID')
```

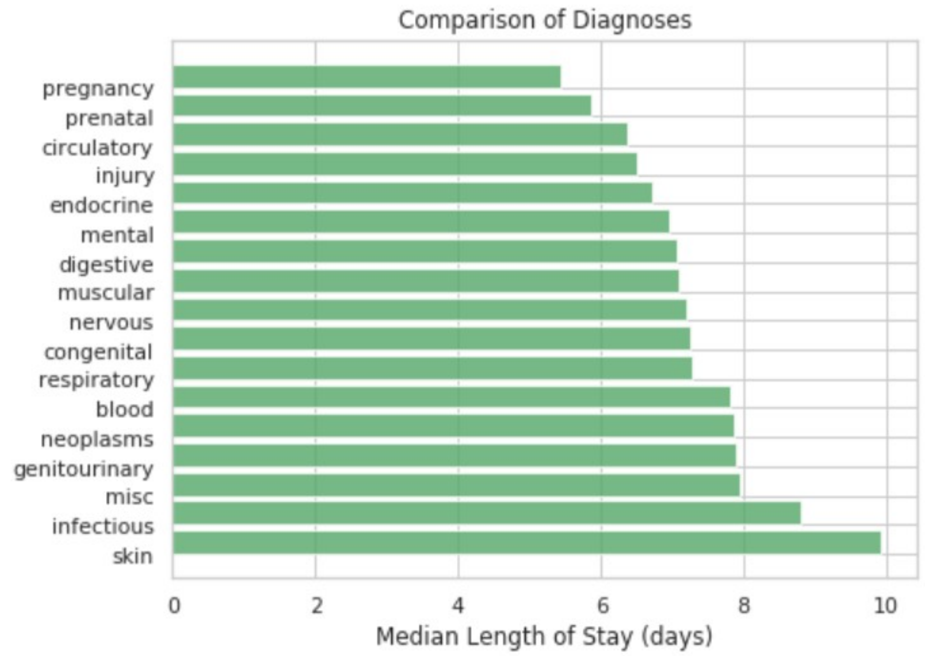
```
# Verify Merge
df.head()
```

IGION	MARITAL_STATUS	ETHNICITY	EDREGTIME	EDOUTTIME	DIAGNOSIS	HOSPITAL_EXPIRE_FLAG	HAS_CHARTEVENTS_DATA	blood	circulatory
IABLE	MARRIED	WHITE	2196-04-09 10:06:00	2196-04-09 13:24:00	BENZODIAZEPINE OVERDOSE	0	1	0	1
HOLIC	MARRIED	WHITE	NaN	NaN	CORONARY ARTERY DISEASE\CORONARY ARTERY BYPASS...	0	1	0	4
HOLIC	MARRIED	WHITE	NaN	NaN	BRAIN MASS	0	1	0	2

```
[58] # Verify Merge
     df.head()
```

	SUBJECT_ID	HADM_ID	ADMITTIME	DEATHTIME	ADMISSION_TYPE	ADMISSION_LOCATION
0	22	165315	2196-04-09 12:26:00	NaN	EMERGENCY	EMERGENCY ROOM ADMIT
1	23	152223	2153-09-03 07:15:00	NaN	ELECTIVE	PHYS REFERRAL/NORMAL DELI
2	23	124321	2157-10-18 19:34:00	NaN	EMERGENCY	TRANSFER FROM HOSP/EXTRAM
3	24	161859	2139-06-06 16:14:00	NaN	EMERGENCY	TRANSFER FROM HOSP/EXTRAM
4	25	129635	2160-11-02 02:06:00	NaN	EMERGENCY	EMERGENCY ROOM ADMIT

```python
# Look at the median LOS by diagnosis category
diag_cat_list = ['skin', 'infectious',  'misc', 'genitourinary', 'neoplasms', 'blood', 'respiratory',
                  'congenital','nervous', 'muscular', 'digestive', 'mental', 'endocrine', 'injury',
                 'circulatory', 'prenatal',  'pregnancy']
results = []
for variable in diag_cat_list:
    results.append(df[[variable, 'LOS']].groupby(variable).median().reset_index().values[1][1])
sns.set(style="whitegrid")
fig, ax = plt.subplots(figsize=(7,5))
ind = range(len(results))
ax.barh(ind, results, align='edge', color = '#55a868', alpha=0.8)
ax.set_yticks(ind)
ax.set_yticklabels(diag_cat_list)
ax.set_xlabel('Median Length of Stay (days)')
ax.tick_params(left=False, right=False, top=False)
ax.set_title('Comparison of Diagnoses'.format(variable))
plt.show()
```



Patients.csv Exploration

- The PATIENTS table provides age and gender information. To protect identity, a patient’s age is given by the difference between their 'DOB' date of birth and the date of their first admission. Therefore, subsequent admissions for the same patient need to be ignored in the calculation. The only things that need to be done with this table are to extract the DOB and gender information and merge them with the admissions dataframe.

```python
df_pat.head()
```

	ROW_ID	SUBJECT_ID	GENDER	DOB	DOD	DOD_HOSP	DOD_SSN	EXPIRE_FLAG
0	234	249	F	2075-03-13 00:00:00	NaN	NaN	NaN	0
1	235	250	F	2164-12-27 00:00:00	2188-11-22 00:00:00	2188-11-22 00:00:00	NaN	1
2	236	251	M	2090-03-15 00:00:00	NaN	NaN	NaN	0
3	237	252	M	2078-03-06 00:00:00	NaN	NaN	NaN	0
4	238	253	F	2089-11-26 00:00:00	NaN	NaN	NaN	0

```python
df_pat['GENDER'].value_counts()
```

```
M     26121
F     20399
Name: GENDER, dtype: int64
```

```python
[114] # Convert to datetime type
df_pat['DOB'] = pd.to_datetime(df_pat['DOB'])
```

```python
df_pat = df_pat[['SUBJECT_ID', 'DOB', 'GENDER']]
df_pat.head()
```

	SUBJECT_ID	DOB	GENDER
0	249	2075-03-13	F
1	250	2164-12-27	F
[illegible]	251	2090-03-15	M
[illegible]	252	2078-03-06	M
[illegible]	253	2089-11-26	F

```python
df = df.merge(df_pat, how='inner', on='SUBJECT_ID')
```

age

```python
# Find the first admission time for each patient
df_age_min = df[['SUBJECT_ID', 'ADMITTIME']].groupby('SUBJECT_ID').min().reset_index()
df_age_min.columns = ['SUBJECT_ID', 'ADMIT_MIN']
df_age_min.head()
```

	SUBJECT_ID	ADMIT_MIN
0	2	2138-07-17 19:04:00
1	3	2101-10-20 19:08:00
2	4	2191-03-16 00:28:00
3	5	2103-02-02 04:31:00
4	6	2175-05-30 07:15:00

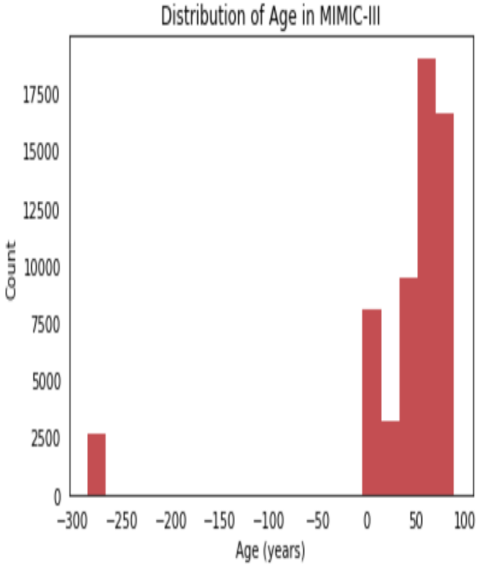
```python
# Verify merge
df.head()
```

	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	DEATHTIME
0	21	22	165315	2196-04-09 12:26:00	2196-04-10 15:54:00	NaN
1	22	23	152223	2153-09-03 07:15:00	2153-09-08 19:10:00	NaN
2	23	23	124321	2157-10-18 19:34:00	2157-10-25 14:00:00	NaN
3	24	24	161859	2139-06-06 16:14:00	2139-06-09 12:48:00	NaN

```
df['ADMIT_MIN'] = pd.to_datetime(df['ADMIT_MIN'])
df['DOB'] = pd.to_datetime(df['DOB'])
df['age'] = (df['ADMIT_MIN'].sub(df['DOB'], axis=0))// np.timedelta64(1, 'Y')
print(df.head())
df['age'].isnull().sum()
```

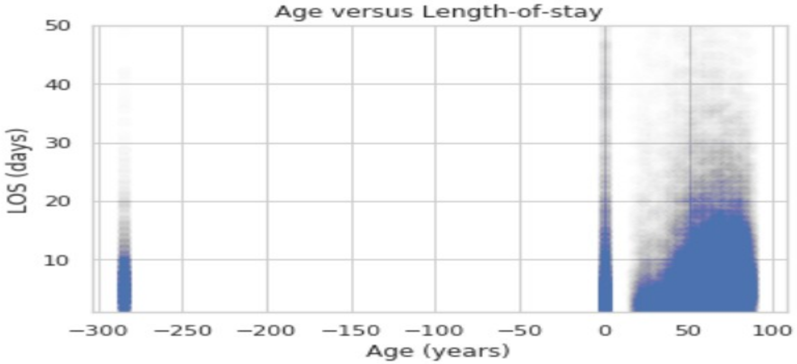
```
   ROW_ID  SUBJECT_ID  HADM_ID  ... GENDER  ADMIT_MIN age
0      21          22   165315  ...      F 2196-04-09  64
1      22          23   152223  ...      M 2153-09-03  71
2      23          23   124321  ...      M 2153-09-03  71
3      24          24   161859  ...      M 2139-06-06  39
4      25          25   129635  ...      M 2160-11-02  58

[5 rows x 40 columns]
0
```

```
# Note that no 'middle' patients show up - this reflects the fact that MIMIC-III does not contain data from pediatric patients.
plt.hist(df['age'], bins=20, color='#c44e52')
plt.ylabel('Count')
plt.xlabel('Age (years)')
plt.title('Distribution of Age in MIMIC-III')
plt.tick_params(left=False, bottom=False, top=False, right=False)
plt.show();
```




```
plt.scatter(df['age'], df['LOS'], alpha=0.005)
#plt.yscale('sqrt')
plt.ylabel('LOS (days)')
plt.xlabel('Age (years)')
plt.title('Age versus Length-of-stay')
plt.ylim(1, 50)
```

```
(1, 50)
```

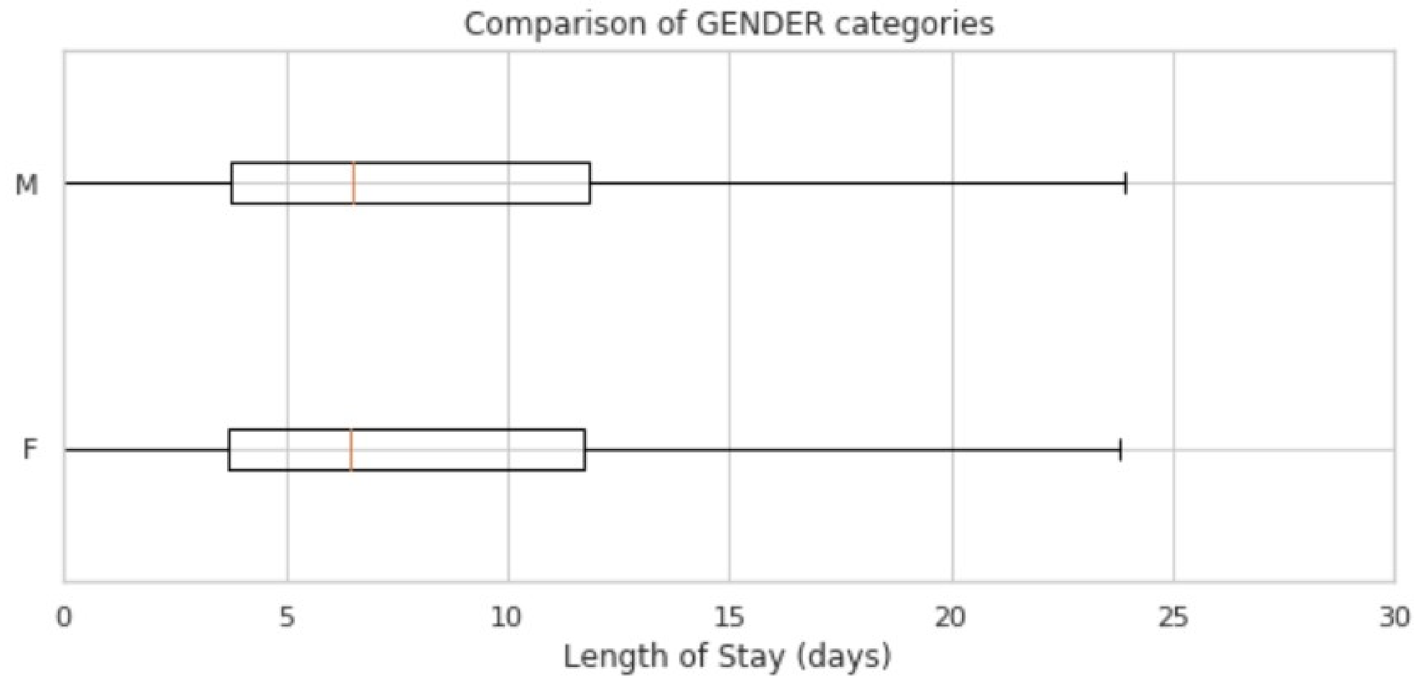


```python
# https://en.wikipedia.org/wiki/List_of_ICD-9_codes
age_ranges = [(0, 13), (13, 36), (36, 56), (56, 100)]
for num, cat_range in enumerate(age_ranges):
    df['age'] = np.where(df['age'].between(cat_range[0],cat_range[1]),
            num, df['age'])

age_dict = {0: 'newborn', 1: 'young_adult', 2: 'middle_adult', 3: 'senior'}
df['age'] = df['age'].replace(age_dict)
df.age.value_counts()
```

```
senior          31188
middle_adult    12781
newborn          8110
young_adult      4281
-285             2616
Name: age, dtype: int64
```

Gender

```
boxplot_los_groupby('GENDER', los_range=(0, 30))
df['GENDER'].replace({'M': 0, 'F':1}, inplace=True)
```



ICUSTAYS.csv Explo

```
df_icu['HADM_ID'].nunique()
```

```
57786
```

```
df_icu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61532 entries, 0 to 61531
Data columns (total 12 columns):
ROW_ID            61532 non-null int64
SUBJECT_ID        61532 non-null int64
HADM_ID           61532 non-null int64
ICUSTAY_ID        61532 non-null int64
DBSOURCE          61532 non-null object
FIRST_CAREUNIT    61532 non-null object
LAST_CAREUNIT     61532 non-null object
FIRST_WARDID      61532 non-null int64
LAST_WARDID       61532 non-null int64
INTIME            61532 non-null object
OUTTIME           61522 non-null object
LOS               61522 non-null float64
dtypes: float64(1), int64(6), object(5)
memory usage: 5.6+ MB
```

```
df_icu.groupby('FIRST_CAREUNIT').median()
```

	ROW_ID	SUBJECT_ID	HADM_ID	ICUSTAY_ID	FIRST_WARDID	LAST_WARDID	LOS
FIRST_CAREUNIT							
CCU	29091.5	22964.5	150074.5	249373.5	7.0	7.0	2.19775
CSRU	31002.5	24488.0	150225.0	250492.0	14.0	14.0	2.15290
MICU	33612.5	26489.5	150368.0	250524.0	50.0	50.0	2.09550
NICU	19581.5	15456.5	149206.5	249308.0	56.0	56.0	0.80250
SICU	38089.0	30084.0	149744.0	248649.0	33.0	33.0	2.25220
TSICU	36382.0	28716.0	148915.0	250685.0	14.0	14.0	2.11150

```
[168] # Based on above statistics, reduce to just ICU and NICU groups
df_icu['FIRST_CAREUNIT'].replace({'CCU': 'ICU', 'CSRU': 'ICU', 'MICU': 'ICU',
                                  'SICU': 'ICU', 'TSICU': 'ICU'}, inplace=True)
```

```
df_icu['cat'] = df_icu['FIRST_CAREUNIT']
icu_list = df_icu.groupby('HADM_ID')['cat'].apply(list).reset_index()
icu_list.head()
```

	HADM_ID	cat
0	100001	[ICU]
1	100003	[ICU]
2	100006	[ICU]
3	100007	[ICU]
4	100009	[ICU]

```
# Create admission-ICU matrix
icu_item = pd.get_dummies(icu_list['cat'].apply(pd.Series).stack()).sum(level=0)
icu_item[icu_item >= 1] = 1
icu_item = icu_item.join(icu_list['HADM_ID'], how="outer")
icu_item.head()
```

	ICU	NICU	HADM_ID
0	1	0	100001
1	1	0	100003
2	1	0	100006
3	1	0	100007
4	1	0	100009

```
df_icu['FIRST_CAREUNIT'].value_counts()
```

```
ICU     53432
NICU     8100
Name: FIRST_CAREUNIT, dtype: int64
```

```python
print("Number of admissions to ICU {}.".format(icu_item.ICU.sum()))
print("Number of admissions to NICU {}.".format(icu_item.NICU.sum()))
```

```
Number of admissions to ICU 49794.
Number of admissions to NICU 7992.
```

[173]

```python
# Merge ICU data with main dataFrame
df = df.merge(icu_item, how='outer', on='HADM_ID')
```

```python
# Replace NaNs with 0
df['ICU'].fillna(value=0, inplace=True)
df['NICU'].fillna(value=0, inplace=True)
```

```python
# Verify NaN fix
print(df.ICU.value_counts(dropna=False))
print(df.NICU.value_counts(dropna=False))
```

```
1.0    49794
0.0     9182
Name: ICU, dtype: int64
0.0    50984
1.0     7992
Name: NICU, dtype: int64
```

Data Preprocessing

- Even after completing the feature engineering for age and ICD-9, there were some loose ends that needed tidying up before the data could be used for the prediction model. First, I ensured that no admissions resulting in death were part the cleaned dataset. I dropped all unused columns and verified that no NaNs existed in the data. For the admission type, insurance type, religion, ethnicity, age, and marital status columns, I performed the Pandas get_dummies command to convert these categorical variables into dummy/indicator variables. The final DataFrame size resulted in 48 feature columns and 1 target column with an entry count of 53,104.

```python
# Remove deceased persons as they will skew LOS result
df = df[df['DECEASED'] == 0]

# Remove LOS with negative number, likely entry form error
df = df[df['LOS'] > 0]
```

```python
# Drop unused or no longer needed columns
df.drop(columns=['SUBJECT_ID', 'HADM_ID', 'ADMITTIME', 'ADMISSION_LOCATION',
                 'DISCHARGE_LOCATION', 'LANGUAGE', 'ADMIT_MIN', 'DOB',
                 'DIAGNOSIS', 'DECEASED',  'DEATHTIME'], inplace=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53104 entries, 0 to 58877
Data columns (total 27 columns):
ADMISSION_TYPE    53104 non-null object
INSURANCE         53104 non-null object
RELIGION          53104 non-null object
MARITAL_STATUS    53104 non-null object
ETHNICITY         53104 non-null object
LOS               53104 non-null float64
blood             53104 non-null float64
circulatory       53104 non-null float64
congenital        53104 non-null float64
```

```python
# Create dummy columns for categorical variables
prefix_cols = ['ADM', 'INS', 'REL', 'ETH', 'AGE', 'MAR', 'RELIGION']
dummy_cols = ['ADMISSION_TYPE', 'INSURANCE', 'RELIGION',
             'ETHNICITY', 'age', 'MARITAL_STATUS', 'RELIGION']
df = pd.get_dummies(df, prefix=prefix_cols, columns=dummy_cols)
df.info()
```

```
Int64Index: 53104 entries, 0 to 58877
Data columns (total 53 columns):
LOS                              53104 non-null float64
blood                            53104 non-null float64
circulatory                      53104 non-null float64
congenital                       53104 non-null float64
digestive                        53104 non-null float64
endocrine                        53104 non-null float64
genitourinary                    53104 non-null float64
infectious                       53104 non-null float64
injury                           53104 non-null float64
mental                           53104 non-null float64
misc                             53104 non-null float64
muscular                         53104 non-null float64
neoplasms                        53104 non-null float64
nervous                          53104 non-null float64
pregnancy                        53104 non-null float64
```

```python
# Verify
df.head()
```

	LOS	blood	circulatory	congenital	digestive	endocrine	genitourinary	infectious
0	1.144444	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1	5.496528	0.0	4.0	0.0	0.0	1.0	1.0	0.0
2	6.768056	0.0	2.0	0.0	0.0	2.0	0.0	0.0
3	2.856944	0.0	2.0	0.0	1.0	1.0	0.0	0.0
4	3.534028	0.0	3.0	0.0	0.0	1.0	0.0	0.0

Data-clean function

- Put all the above into one data clean function

```
def mimic_los_cleanup(adm_csv='ADMISSIONS.csv', patients_csv='PATIENTS.csv',
                      diagcode_csv='DIAGNOSES_ICD.csv', icu_csv='ICUSTAYS.csv',
                      verbose=True):
    '''
    This function take 4 csv files from the MIMIC-III database, converts them to
    DataFrames for cleanup and feature engineering for use in a Length-of-Stay
    regression model such as the sklearn GradientBoostingRegressor.

    INPUT:
    adm_csv - Primary Admissions information
    patients_csv - Patient specific info such as gender and DOB
    diagcode_csv - ICD9 Diagnosis for each admission to hospital
    icu_csv - Intensive Care Unit (ICU) data for each admission

    OUTPUT:
    df - clean DataFrame for use in an regression model
    actual_median_los - Median LOS for all admissions
    actual_mean_los - Average LOS for all admissions
```

Length-of-Stay Prediction Model

- To implement the prediction model, I split the LOS target variable and features into training and test sets at an 80:20 ratio using the scikit-learn train_test_split function. Using the training set, I'll fit five different regression models (from the scikit-learn library) using default settings to see what the R2 score comparison looked like.

```
# Target Variable (Length-of-Stay)
LOS = df['LOS'].values
# Prediction Features
features = df.drop(columns=['LOS'])
```

```python
# Split into train 80% and test 20%
X_train, X_test, y_train, y_test = train_test_split(features,
                                                    LOS,
                                                    test_size = .20,
                                                    random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

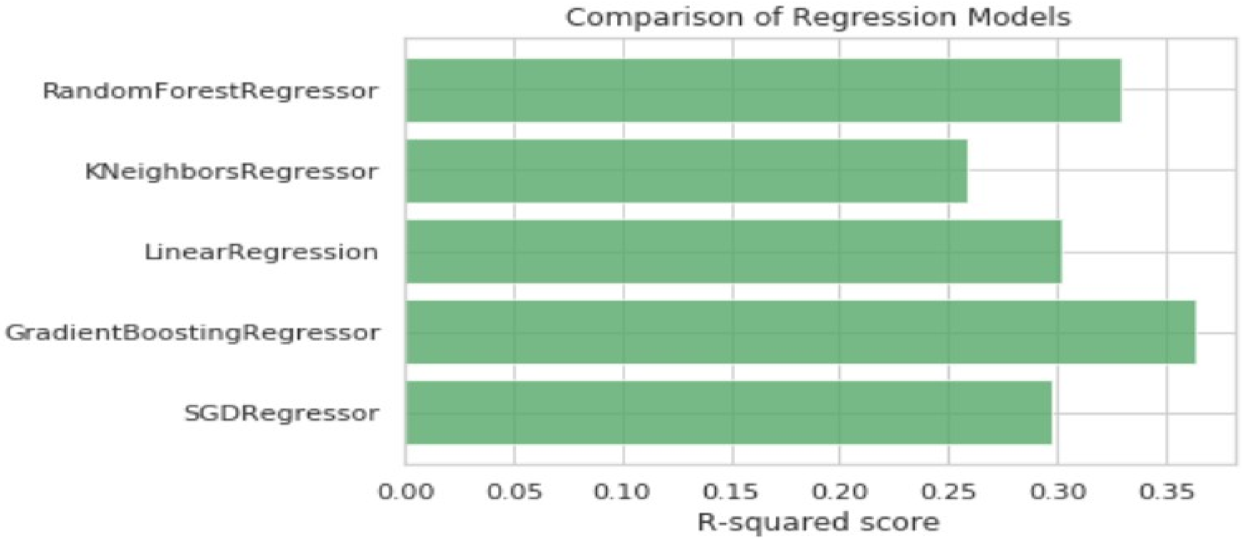
```
Training set has 42483 samples.
Testing set has 10621 samples.
```

```python
# Regression models for comparison
models = [SGDRegressor(random_state = 0),
          GradientBoostingRegressor(random_state = 0),
          LinearRegression(),
          KNeighborsRegressor(),
          RandomForestRegressor(random_state = 0)]

results = {}

for model in models:

    # Instantiate and fit Regressor Model
    reg_model = model
    reg_model.fit(X_train, y_train)

    # Make predictions with model
    y_test_preds = reg_model.predict(X_test)

    # Grab model name and store results associated with model
    name = str(model).split("(")[0]
```

```
SGDRegressor done.
GradientBoostingRegressor done.
LinearRegression done.
KNeighborsRegressor done.
RandomForestRegressor done.
```

```python
# R2 score results
fig, ax = plt.subplots()
ind = range(len(results))
ax.barh(ind, list(results.values()), align='center',
        color = '#55a868', alpha=0.8)
ax.set_yticks(ind)
ax.set_yticklabels(results.keys())
ax.set_xlabel('R-squared score')
ax.tick_params(left=False, top=False, right=False)
ax.set_title('Comparison of Regression Models')
#fig.savefig('images/compare_models.png', bbox_inches = 'tight')
```

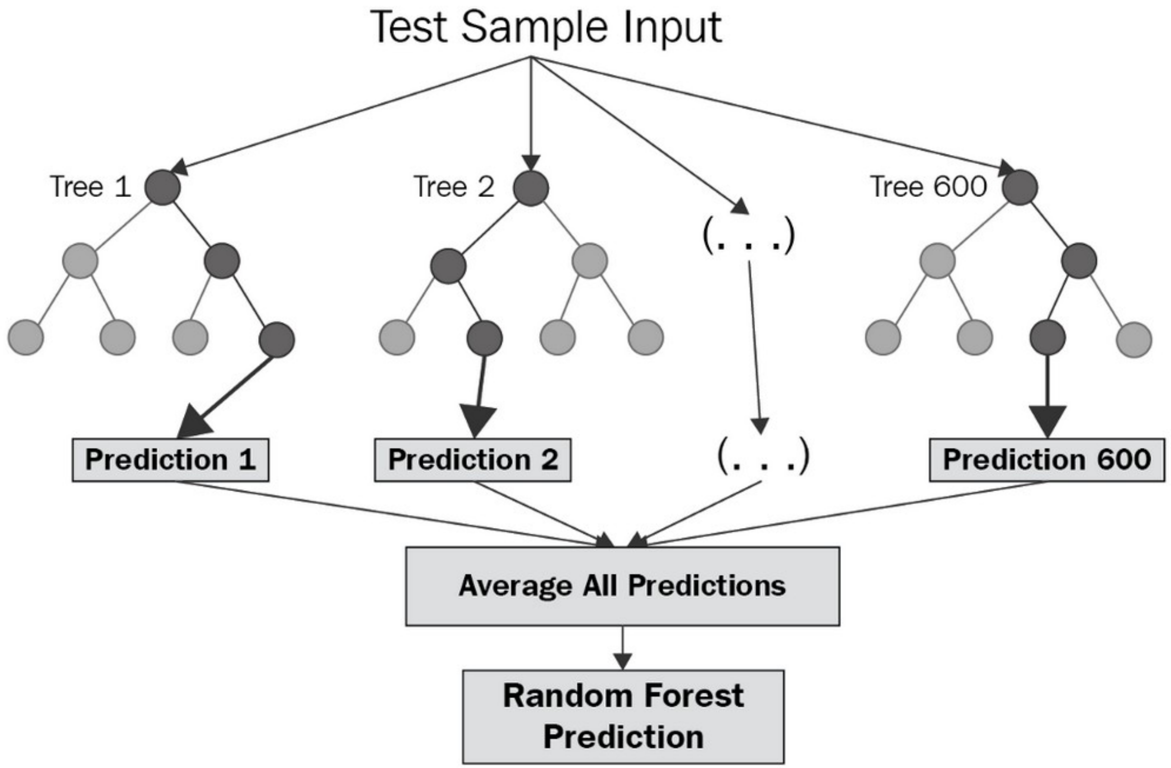
```
Text(0.5, 1.0, 'Comparison of Regression Models')
```




```python
# GradientBoostingRegressor will be used as the LOS prediction model
reg_model = GradientBoostingRegressor(random_state=0)
reg_model.fit(X_train, y_train)
y_test_preds = reg_model.predict(X_test)
r2_not_refined = r2_score(y_test, y_test_preds)
print("R2 score is: {:2f}".format(r2_not_refined))
```

```
R2 score is: 0.363717
```

Model: Random Forest



Random Forest Structure

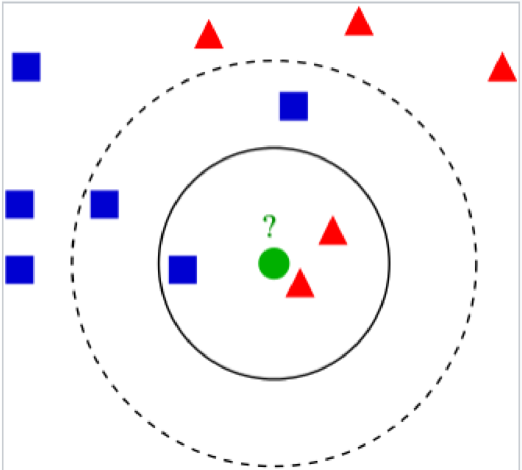
- Random Forest is an ensemble learning method for classification, regression

- Random Forest is a bagging technology that each tree runs parallel and does not communicate with each other. The final result is the average of predictions from all trees (bagging)

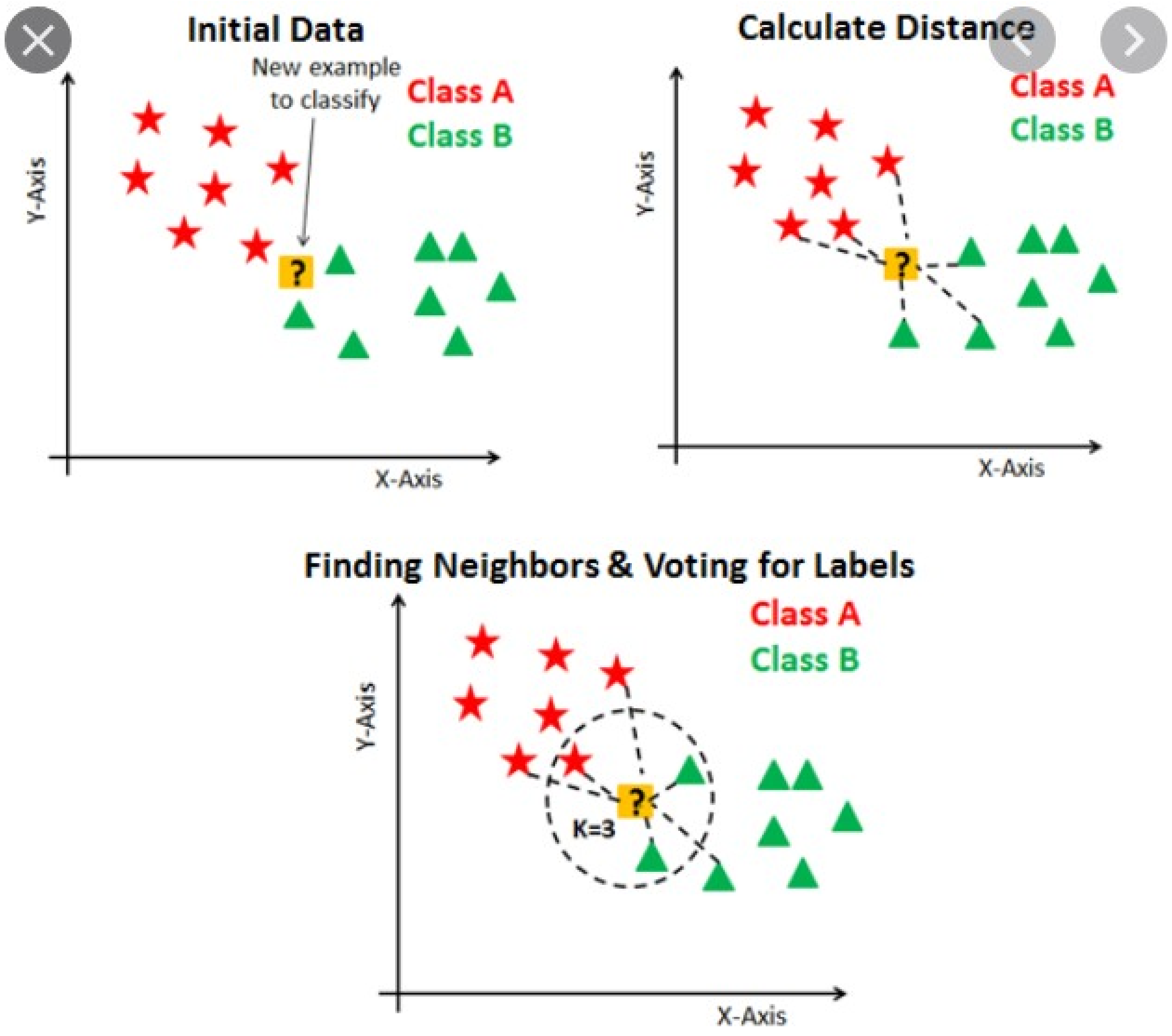
- It is an ensemble solution which aggregates the decision from each tree.

- It is a highly accurate classifier

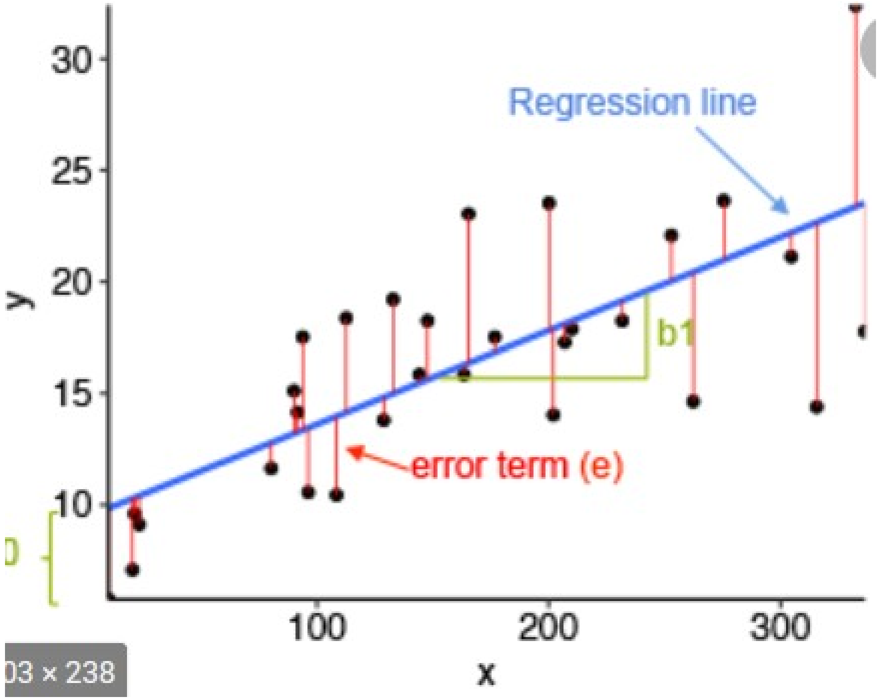
Model: K-Nearest Neighbors (KNN)

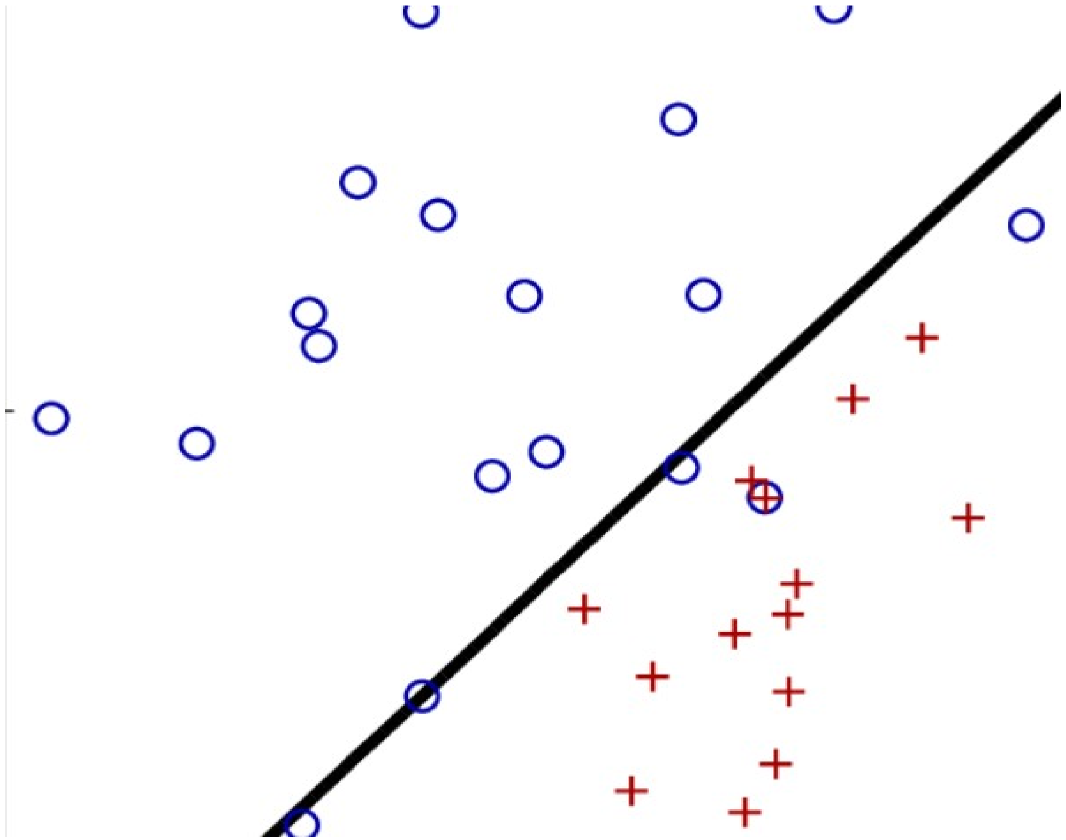


Example of *k*-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If *k* = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If *k* = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

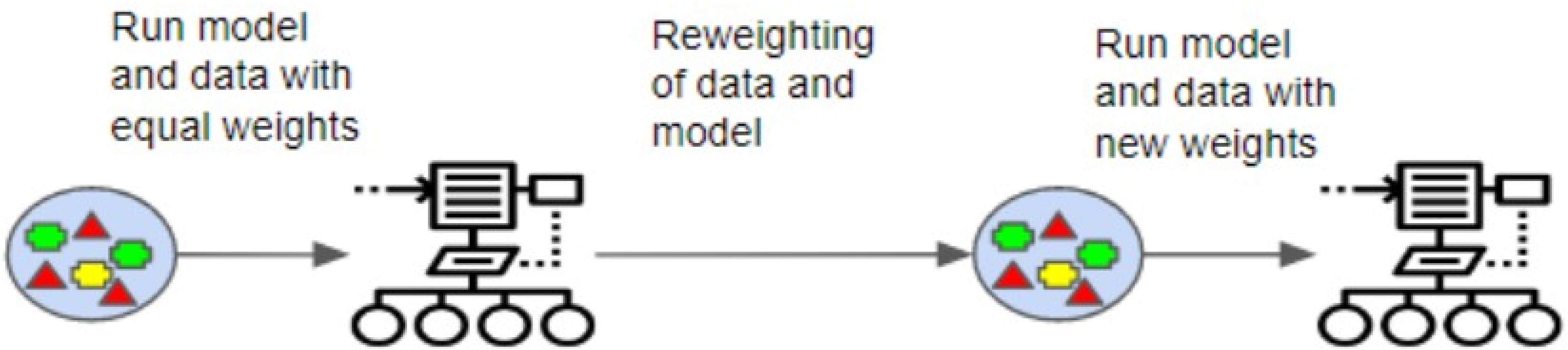


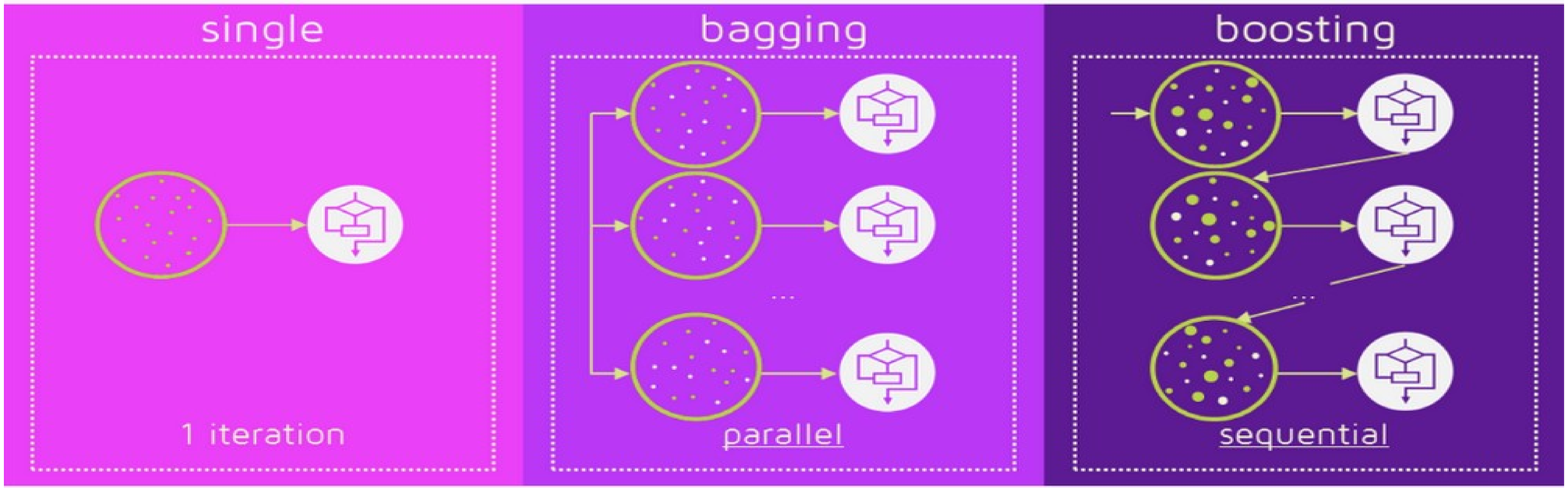
Model: Linear Regression



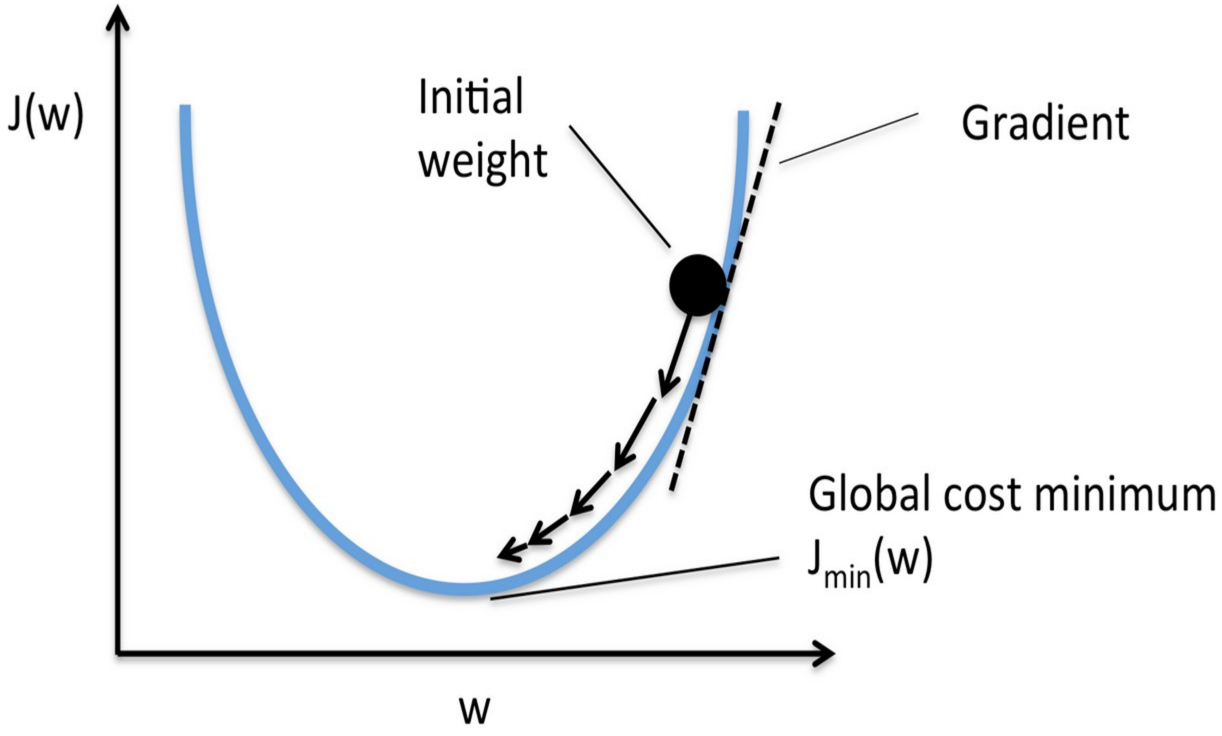


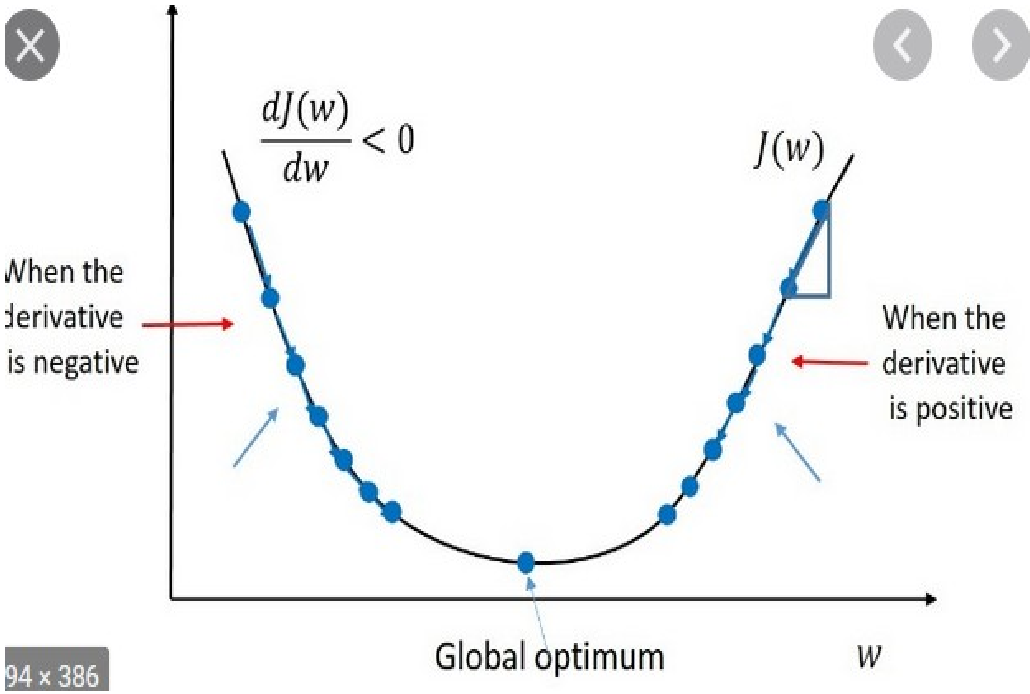
Model: Gradient Boosting Classifier



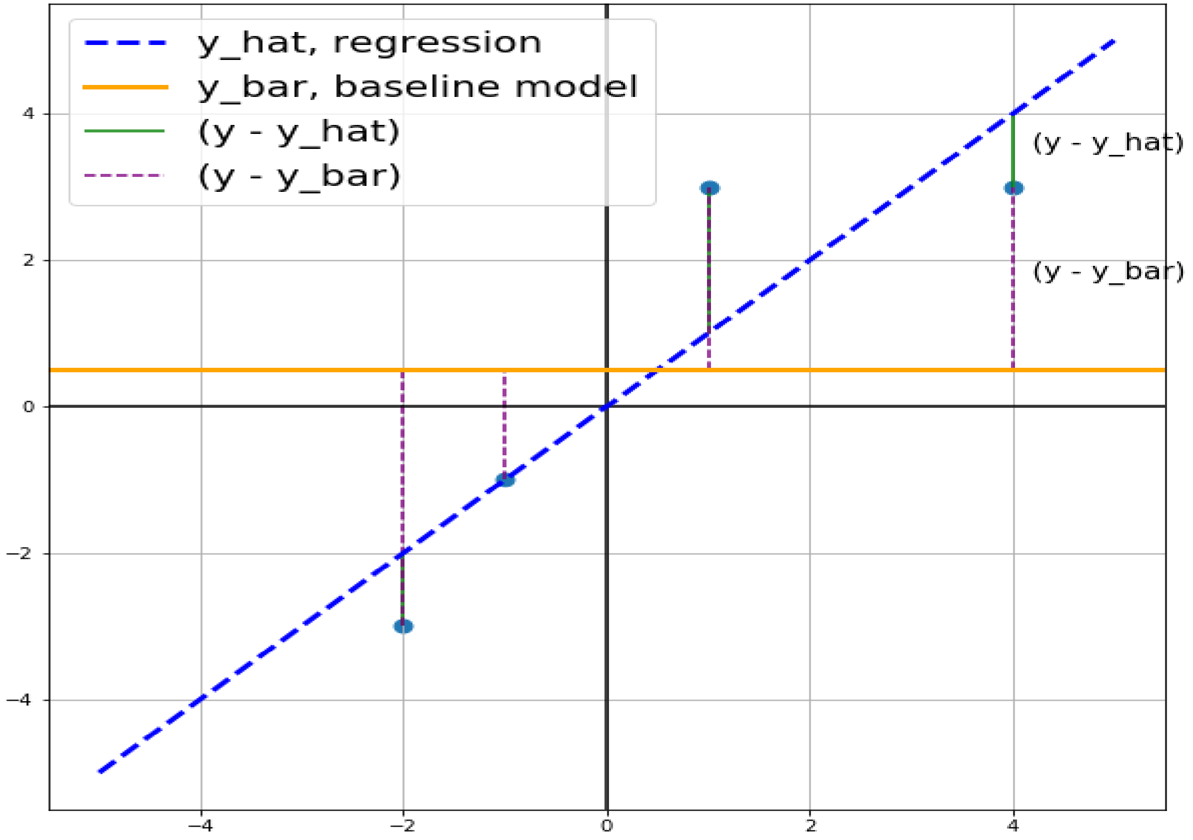


Model: Stochastic Gradient Descent





$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2}$$



Model Refinement

- The GradientBoostingRegressor performed well versus the other regression model. To refine the GradientBoostingRegressor model, I used the GridSearchCV function from scikit-learn to test out various permutations of parameters such as n_estimators, max_depth, and loss. The best estimator result from GridSearchCV was n_estimators=200, max_depth=4, and loss=ls.

```python
# Split into train 80% and test 20%
X_train, X_test, y_train, y_test = train_test_split(features,
                                                    LOS,
                                                    test_size = .20,
                                                    random_state = 42)

# Set the parameters by cross-validation
#tuned_parameters = [{'n_estimators': [100, 200, 300],
#                      'max_depth' : [2, 3, 4],
#                      'loss': ['ls', 'lad', 'huber']}]
tuned_parameters = [{'n_estimators': [200, 300],
                     'max_depth' : [3, 4],
                     'loss': ['ls', 'lad']}]

# create and fit a ridge regression model, testing each alpha
reg_model = GradientBoostingRegressor()
grid = GridSearchCV(reg_model, tuned_parameters)
grid.fit(X_train, y_train)
reg_model_optimized = grid.best_estimator_
```

Least Absolute Deviations (LAD)
Least Squares (LS)

```
0.4105185082515508
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=4,
                          max_features=None, max_leaf_nodes=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=200,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

$$R^2 \equiv 1 - \frac{\sum_i (y_i - \bar{y})^2}{\sum_i (y_i - f_i)^2}$$

```
#reg_model = GradientBoostingRegressor(n_estimators = 200, max_depth=4, random_state=0)
#reg_model.fit(X_train, y_train)
y_test_preds = reg_model_optimized.predict(X_test)
r2_optimized = r2_score(y_test, y_test_preds)
print("Optimized R2 score is: {:2f}".format(r2_optimized))
```

```
Optimized R2 score is: 0.387081
```

```
print('Model refinement improved R2 score by {:.4f}'.format(r2_optimized-r2_not_refined))
```

```
Model refinement improved R2 score by 0.0234
```

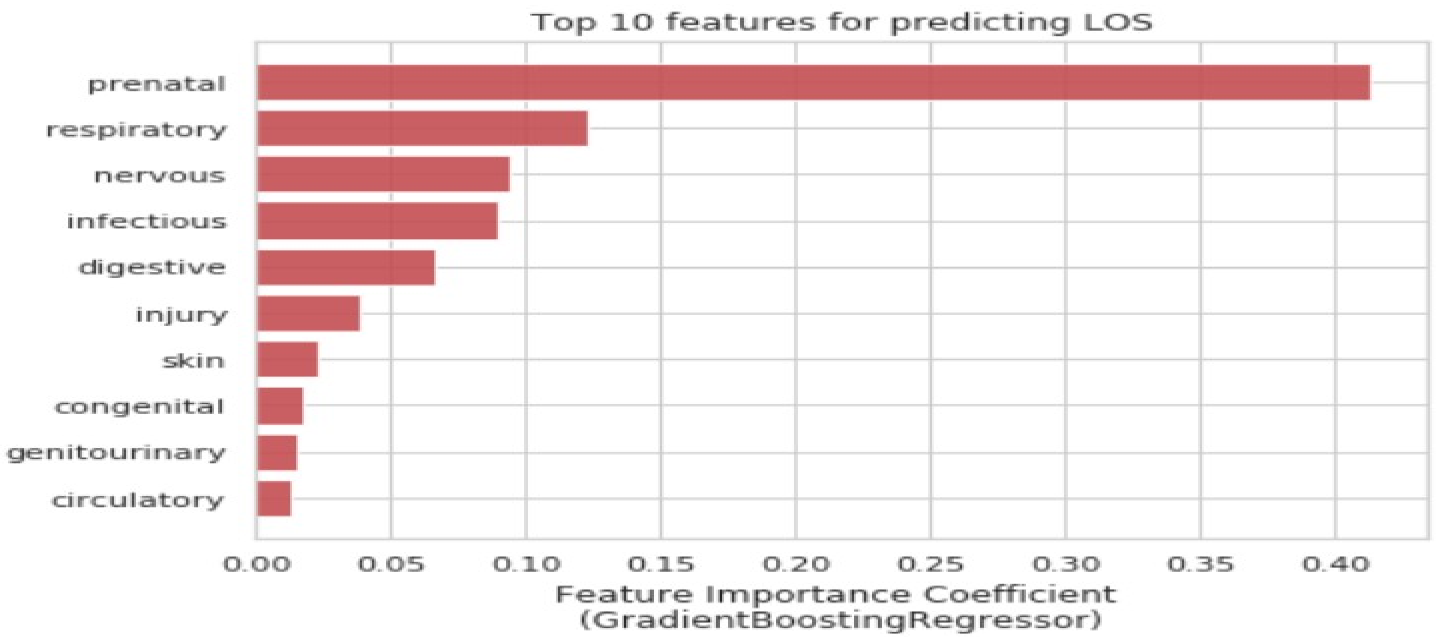
Evaluation

- Feature Importance
  - In fact, in the top 20 top features, only emergency admission type, gender, and Medicaid insurance showed any importance outside of diagnoses groups.

```
# https://towardsdatascience.com/running-random-forests-inspect-the-feature-importances-with-this-code-2b00dd72b92e
feature_imp = pd.DataFrame(reg_model_optimized.feature_importances_,
                           index = X_train.columns,
                           columns=['importance']).sort_values('importance', ascending=False)


feature_imp.head(20)
```

	importance
prenatal	0.413576
respiratory	0.123059
nervous	0.093692
infectious	0.089346
digestive	0.066122
injury	0.038014

```python
# Plot feature importance
fig, ax = plt.subplots(figsize=(7, 5))
ind = range(0,10)
ax.barh(ind, feature_imp['importance'].values[0:10],
        align='center', color='#c44e52', alpha=0.9)
ax.set_yticks(ind)
ax.set_yticklabels(feature_imp.index[0:10].tolist())
ax.tick_params(left=False, top=False, right=False)
ax.set_title("Top 10 features for predicting LOS")
ax.set_xlabel('Feature Importance Coefficient \n(GradientBoostingRegressor)')
plt.gca().invert_yaxis()
#fig.savefig('images/feature_importance.png', bbox_inches = 'tight')
```



Evaluation

```
actual_avg_los = df['LOS'].mean()
ml_count, md_count, avg_count = 0, 0, 0
ml_days, md_days, avg_days = 0, 0, 0
ml_days_rms, md_days_rms, avg_days_rms = 0, 0, 0

for i in range(y_test_preds.shape[0]):
    ml_model = abs(y_test_preds[i] - y_test[i])
    median_model = abs(actual_median_los - y_test[i])
    average_model = abs(actual_avg_los - y_test[i])

    ml_days += ml_model
    md_days += median_model
    avg_days += average_model

    ml_model_rms = (y_test_preds[i] - y_test[i]) ** 2
    median_model_rms = (actual_median_los - y_test[i]) ** 2
    average_model_rms = (actual_avg_los - y_test[i]) ** 2
```

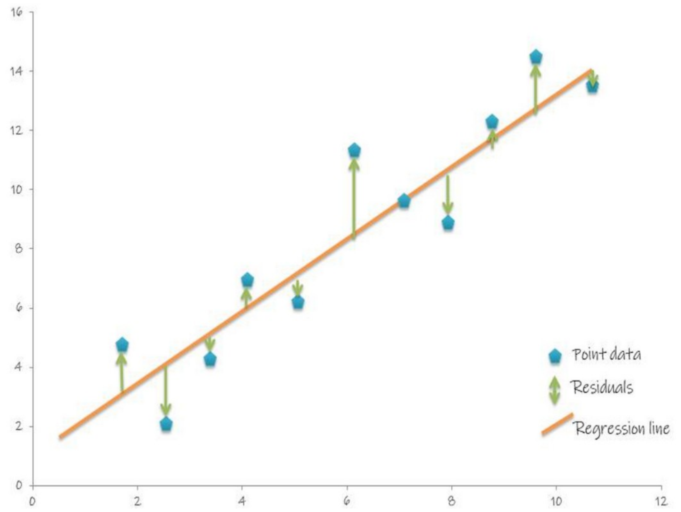
Evaluation

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

Lower **values** of **RMSE** indicate **better** fit

The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values.

RMSE is a good measure of how accurately the model predicts the response, and it is the most important criterion for fit if the main purpose of the model is prediction



a lower RMSD is better than a higher one

```
    ml_days_rms += ml_model_rms
    md_days_rms += median_model_rms
    avg_days_rms += average_model_rms

print("Prediction Model days {}".format(ml_days / y_test_preds.shape[0]))
print("Median Model days {}".format(md_days / y_test_preds.shape[0]))
print("Average Model days {}".format(avg_days / y_test_preds.shape[0]))

print("Prediction Model RMS {}".format((ml_days_rms ** 0.5) / y_test_preds.shape[0]))
print("Median Model RMS {}".format((md_days_rms ** 0.5) / y_test_preds.shape[0]))
print("Average Model RMS {}".format((avg_days_rms ** 0.5) / y_test_preds.shape[0]))
# RMSE plot for writeup
data = pd.DataFrame({'RMSE': [(ml_days_rms**0.5)/y_test_preds.shape[0],
                             (avg_days_rms**0.5)/y_test_preds.shape[0],
                             (md_days_rms**0.5)/y_test_preds.shape[0]],
                     'LOS Model Type': ['Predicted LOS', 'Average LOS', 'Median LOS'] })

fig, ax = plt.subplots()
ax = sns.barplot(x='RMSE', y='LOS Model Type', data=data)
ax.set_title('RMSE comparison of Length-of-Stay models')
ax.tick_params(top=False, left=False, right=False)

plt.show()
```

```
Prediction Model days 5.433504530531759
Median Model days 6.387375443304172
Average Model days 7.223816450884783
Prediction Model RMS 0.09573626208240148
Median Model RMS 0.12697346364877565
Average Model RMS 0.12228645096382446
```