PORTOFOLIO

Yubo Wang

Yubo Wang

CONTACT

robertwangyb@outlook.com

647-936-7160

www.linkedin.com/in/robertwyb

https://github.com/robertwyb

SKILLS

Languages

- Python
- MySQL
- C++
- C
- Java
- Javascript
- HTML
- R

Frameworks

- Numpy
- Pandas
- Sklean
- Matplotlib
- Plotly Dash
- OpenCV
- Git
- Linux
- PyTorch
- Tensorflow
- Selenium
- Beautiful Soup
- Flask

Big Data

- Hadoop
- Hive
- Spark
- AWS (EC2, EMR, S3, Redshift)

Tools

- Tableau
- PowerBI

EDUCATION

Degree

Honor Bachelor of Science
in Computer Science and Statistics

University of Toronto St.George
2015 - 2019

Data Science Bootcamp (WeCloudData)

PROJECTS DEMO

Github Repo

http://bit.ly/face-recog

http://bit.ly/lol-winrate-prediction

http://bit.ly/used-car-price-predict

http://bit.ly/deeplearning-assignments

Online Dashboard

https://van-house-price.herokuapp.com/

EXPERIENCE

2019-09 - Present

Data Science Consultant

Beam Data, Toronto, ON

Client: Samsung Electronics Canada
Project: Device Sales Forecasting

- Currently working with the client's Advanced Analytics team leader and data scientists on a new sales forecasting model to help the company improve inventory operations
- Extracted historical sales data from SQL server and prepared data for machine learning
- Implemented dynamic Machine Learning model using sklearn to predict the next four months product distribution based on past sales data and automatically adjust based on current month sales data

Client: Healthcare Client at UBC
Project: Worked with medical doctors and researchers on phenomapping of patients with Heart Failure with preserved Ejection Fraction cluster analysis

- Cleaned messy patients data using Python and Pandas
- Implemented K-Means clustering models to find multiple clusters of patients having HFpEF based on patients' biometrics and medical history

PROJECTS

2019-09 - 2019-10

Loan Default Prediction

- Loaded 8 files into MySQL database, joined and cleaned data for machine learning model training, preprocessed and analyzed data using Pandas
- Implemented machine learning model using several algorithms (Logistic Regression, Decision Tree, Random Forest, Gradient Boosting)
- Used Cross Validation and SMOTE to handle imbalanced data
- Used Grid Search and Pipeline to tune hyperparameters to enhance the recall and F1 score from 0.53 to 0.75 for predicting defaulted loan

2018-11 - 2018-12

Face recognition and scene reconstruction

Github: *https://github.com/robertwyb/face-recognition-project*

- Implemented panorama stitching algorithm using feature matching and SIFT
- Trained classifier using KNN and SVM to detect faces, label them as male, female and predict age

2019-10 - 2019-11

Realtime gamea(League of Legends) winrate prediction

Github: *https://github.com/robertwyb/League-of-Legends-winrate-prediction*

- Gathered most recent 10 games history of over 400,000 high-level players from Riot official game API
- Consolidated and prepared data to extract useful feature for machine learning model training
- Trained Logistic Regression model to predict win rate before the game starts based on team composition and players' mastery, achieved 0.9 accuracy
- Trained KNN using OpenCV for digit recognition to get in-game player data from live screen capture
- Trained Gradient Boosting model using XGBoost to predict real time win rate based on in-game team and player data

2019-09 - 2019-10

Vancouver House Price Analysis and Prediction

Github: *https://github.com/robertwyb/vancouver-house-price-scraping*
Dashboard: *https://van-house-price.herokuapp.com/*

- Scrapped data using Selenium and Beautiful Soup from house price website to analyze house market in Vancouver.
- Implemented Multi Linear Regression model to predict house price given region, area, year, rooms, and garage, produce reasonable predicted price compared with average prices of models with same condition selling online
- Train KMeans to cluster houses into several groups, try to find is there specific group having higher unit price/price change rate
- Visualized data and model prediction using plotly and created a dashboard for presenting and using model

2019-09 - 2019-10

Deep Learning Assignments

Github: *https://github.com/robertwyb/Neural-Networks-Assignments*

- Used PyTorch to implement CNN for image colorization, Attention-based translation, and Cycle-GAN

TABLE OF CONTENTS

LOAN DEFAULT
PREDICTION

GOALS

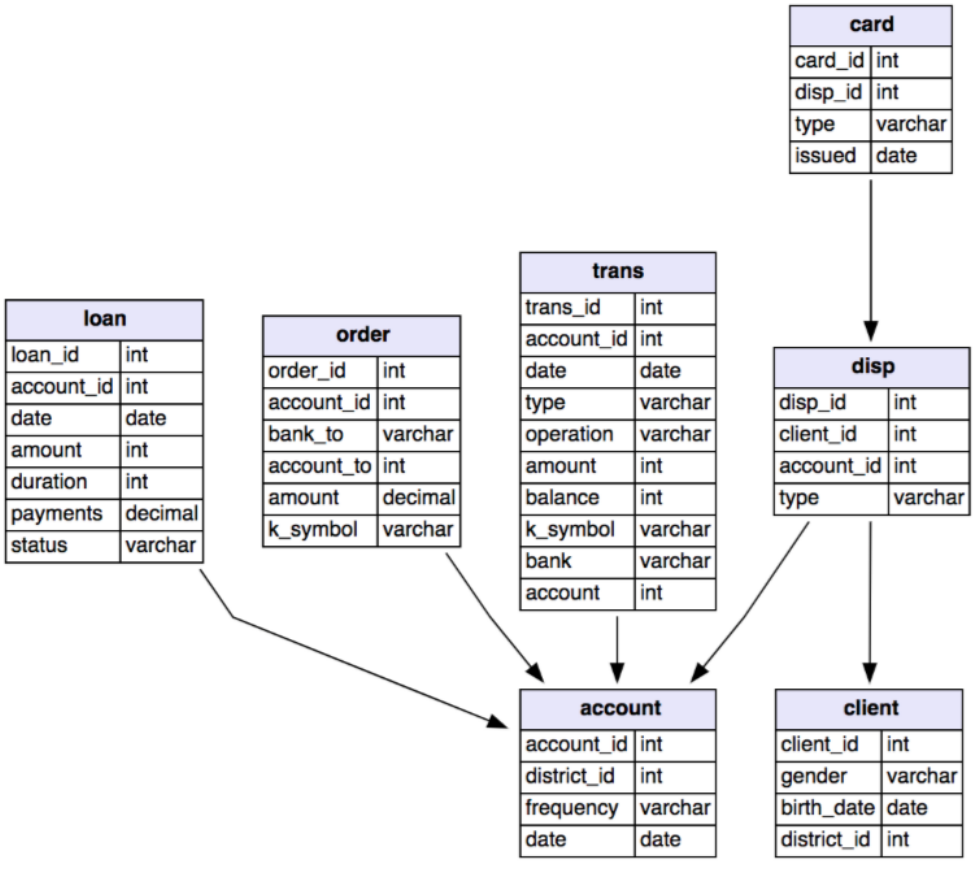
The goal of this project is to build a classification model based on past behavioral data of customers to predict whether the customer will get defaulted or not when they apply for loan so that we can help the financial institution automatically decide whether to approve the loan application for their customers

TOOLS

- MySQL
- Pandas
- Scikit-lean

DATA

- 6 Files stores structured account information and transactions as the followed graph
- 4500 bank accounts with more than 1 million transactions and 700 loan applications
- status in loan table is the target we want to predict using the rest of data



```sql
DROP TABLE IF EXISTS loan;
CREATE TABLE loan (
    loan_id                 INT,
    account_id              INT,
    date                    VARCHAR(50),
    amount                  INT,
    duration                INT,
    payments                DECIMAL(10, 2),
    status                  VARCHAR(50)
);

TRUNCATE loan;

LOAD DATA INFILE './bank_data_loan_default/loan.asc'
INTO TABLE loan
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

Use MySQL to load the raw data

To use the raw data, we need to load them to a databse. I use MySQL to load 6 asc files into 6 tables. Then connect the database to python, using pandas to load each table as DataFrame and preprocess the data using self-defined functions.

```python
# CARD
card['card_year'], card['card_month'], card['card_day'] = split_time(card['issued'])
card['issued'] = pd.Series([i[:6] for i in card['issued']])
card['type'] = card['type'].str.replace('"', '')
card['type'] = convert_card_type(card['type'])

# LOAN
loan['loan_year'], loan['loan_month'], loan['loan_day'] = split_time(loan['date'])
loan['status'] = loan['status'].str.replace('"', '')
# 0 if bad, 1 if good
loan['default'] = convert_loan_status(loan['status'])

# ORDER
order['bank_to'] = order['bank_to'].str.replace('"', '')
order['k_symbol'] = order['k_symbol'].str.replace('"', '')
# k_symbol:0 for null, 1 for insurance, 2 for household, 3 for leasing, 4 for loan
order['order_k_symbol'] = convert_k_symbol(order['k_symbol'])

# TRANS
trans['type'] = trans['type'].str.replace('"', '')
trans['operation'] = trans['operation'].str.replace('"', '')
trans['k_symbol'] = trans['k_symbol'].str.replace('"', '')
trans['bank'] = trans['bank'].str.replace('"', '')
trans['account'] = pd.Series([int(i) if not np.isnan(i) else i for i in trans['account']],  dtype='Int64')
trans['type'] = pd.Series([1 if i == 'PRIJEM' else 0 for i in trans['type']])
trans['operation'] = convert_trans_operation(trans['operation'])
trans['trans_k_symbol'] = convert_k_symbol(trans['k_symbol'])

# DISP
disp['type'] = disp['type'].str.replace('"', '')
disp_owner = disp[disp['type'] == 'OWNER']

# ACCOUNT
account['account_year'], account['account_month'], account['account_day'] = split_time(account['date'])
account['frequency'] = account['frequency'].str.replace('"', '')
# frequency column: 0 for null, 1 for monthly, 2 for weekly, 3 for after transaction
account['frequency'] = convert_account_freq(account['frequency'])

# CLIENT
client['birth_date'] = client['birth_date'].str.replace('"', '')
client['client_birth_year'], client['client_birth_month'], client['client_birth_day'] = \
    split_time(client['birth_date'])
client['client_sex'] = pd.Series([1 if i < 13 else 0 for i in client['client_birth_month']])
client['client_birth_month'] = pd.Series([i if i < 13 else i - 50 for i in client['client_birth_month']])
```

Join tables

Merge 6 dataframes into 1 and extract useful columns before joining to increase join speed

```
# create features, join tables

# get account's order amount
order_df_lst = []
for account_id, group in order.groupby('account_id'):
    order_num = group.shape[0]
    order_sum = group['amount'].sum()
    order_df_lst.append({'account_id': account_id, 'order_num': order_num, 'order_sum': order_sum})
order_df = pd.DataFrame(order_df_lst)

# create final df by merging all tables
card = card.rename(columns={'type': 'card_level'})
df = pd.merge(disp_owner[['disp_id', 'client_id', 'account_id']], card[['disp_id', 'card_level', 'card_year']],
              how='left')
df.card_level = df.card_level.fillna(0).astype(int)
df.card_year = df.card_year.fillna(99).astype(int)

# merge df with client to get gender
df = pd.merge(df, client[['client_id', 'client_sex', 'client_birth_year']])

# merge df with ACCOUNT table
df = pd.merge(df, account[['account_id', 'district_id', 'account_year']])

# merge df with ORDER table
df = pd.merge(df, order_df)

# merge df with LOAN table
df = pd.merge(df, loan[['account_id', 'amount', 'duration', 'payments', 'loan_month', 'loan_day', 'default']],
              how='right')
```

Avoid Data Leakage and feature engineer

To avoid data leakage, only account's information before the loan application date is used as model input.

Based on the balance data, I also create the average balance of this account and ratio of balance before applying loan to loan amount as features to enhance the performance of the model.

```
# get account's balance and income from trans table before loan application to avoid data leak
account_balance_income = []
trans_date_filter = pd.merge(loan, trans, suffixes=('_loan', '_trans'))
trans_date_filter = pd.merge(loan, trans, on='account_id', how='left', suffixes=('_loan', '_trans'))
trans_date_filter = trans_date_filter[trans_date_filter['date_loan'] > trans_date_filter['date_trans']]
for account_id, group in trans_date_filter.groupby('account_id'):
    group.sort_values(by=['trans_id'])
    balance = group['balance'].tail(1).values[0]
    balance_ratio = 1 if balance > group['amount_loan'].iloc[0] * 0.01 else 0
    avg_balance = group['balance'].mean()
    income = group[group['type'] == 1]['amount_trans'].sum()
    trans_count = group.shape[0]

    account_balance_income.append({'account_id': account_id, 'balance': balance, 'balance_ratio': balance_ratio,
                                   'avg_balance': avg_balance, 'income': income, 'trans_count': trans_count
                                   # 'k1': k1, 'k2': k2, 'k3': k3, 'k4': k4, 'k5': k5, 'k6': k6, 'k7': k7
                                   })
trans_df = pd.DataFrame(account_balance_income)

# merge df with TRANS table
df = pd.merge(df, trans_df)
```

MODEL IMPLEMENTING

Algorithm

For this Classification problem, I have tried to use Logistic Regression, K Nearest Neighbours, Decision Tree, Random Forest, XGBoost. To simplify the process of training and evaluating models, I write a function that takes features, target and model that return classification reports.

```
def build_model(X, y, model, model_name):
    """
    build specific model with k-fold
    :param X: train df
    :param y: label df
    :param model: sklearn classifier
    :param model_name: name of the classifier
    :return: dict contain precision, recall and f1 scores
    """
    kf = KFold(n_splits=20, random_state=1, shuffle=True)
    scores = {'precision': [], 'recall': [], 'f1': []}
    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        scaler = StandardScaler()

        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.fit_transform(X_test)

        ros = RandomOverSampler(random_state=42)
        X_train_resampled, y_train_resampled = ros.fit_resample(X_train_scaled, y_train)

        model.fit(X_train_resampled, y_train_resampled)
        y_pred = model.predict(X_test_scaled)
        y_train_pred = model.predict(X_train_scaled)
        scores['precision'].append(precision_score(y_test, y_pred))
        scores['recall'].append(recall_score(y_test, y_pred))
        scores['f1'].append(f1_score(y_test, y_pred))
        print('---------train----------')
        print(classification_report(y_train, y_train_pred))
        print('--------test-----------')
        print(classification_report(y_test, y_pred))

    print('------------------------------------------------------')
    print(model_name)
    p_mean, r_mean, f_mean = np.mean(scores['precision']), np.mean(scores['recall']), np.mean(scores['f1'])
    return {'model': model_name, 'precision score': p_mean, 'recall score': r_mean, 'f1 score': f_mean}
```

Handle Imbalanced Data

Since we are exploring loan defaults data, the ratio of the number of defaults to number of loans are below 10%, which means the dataset we use to train the model will be imbalanced. To fix this problem, I use *RandomOverSampler()* method in *Imblearn* package which randomly samples data points in the minority group until there are same number of points from both groups.

Cross Validation

Although there are more than 4500 bank accounts, only 1000 of them have applied for loan. Therefore, due to lack of data, I use cross validation to enhance the performance of the model, then take the mean score as the final result.

```
models = [
          DecisionTreeClassifier(random_state=1, max_depth=15),
          LogisticRegression(random_state=1, solver='lbfgs'),
          KNeighborsClassifier(n_neighbors=3),
          RandomForestClassifier(n_estimators=200,  random_state=1),
          XGBClassifier()
          ]
scores = []
for m in models:
    score = build_model(X, y, m, str(m)[:str(m).find('(')])
    scores.append(score)
```

Example of using build_model method to train and evaluate models

GridSearch

After briefly checking the performance of each model, although XGBoost has the best F1 score, RandomForest has better recall score, which is a better metric for evaluating loan default model, since recall score punish false-negative more. Therefore, I use GridSearch to get the best hyperparameter for Random Forest.

```
# use gridsearch to tune parameter for RandomForest
parameters = {
    'n_estimators': [80, 90, 100, 110],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': range(2, 100),
    'n_jobs': [-1],
    'class_weight': ['balanced', 'balanced_subsample']
}

model = RandomForestClassifier()
clf = GridSearchCV(model, parameters, cv=5, n_jobs=-1, verbose=2)
clf.fit(X, y)

build_model(clf.best_estimator_)
```

SUMMARY & FUTURE IMPROVEMENT

- Using best hyperparameter we get after using GridSearch, the recall score is 0.75 for predicting defaulted loan
- For future improvement, we can try to create more useful features from the exisiting data

PANORAMA FACIAL RECOGNITION

GOALS

- This project aims at recognizing faces in an environment with several people. A panorama picture will be constructed. Gender, age and distance to camera of all recognized faces will be labeled in the image. From my perspective, this is a quality project to show what we have learned through this semester. This project can be used as tools to produce panorama or wide angle selfie with face tag. And with additional hardware help, it can be an assistance software for blind that can tell them who are surrounding them and how far they are.

TOOLS

- To accomplish this goal, we need to use OpenCV with Python.
- Computer Vision knowledge used: SIFT, Homograph computing, image cutting, Geometry computing

DATA

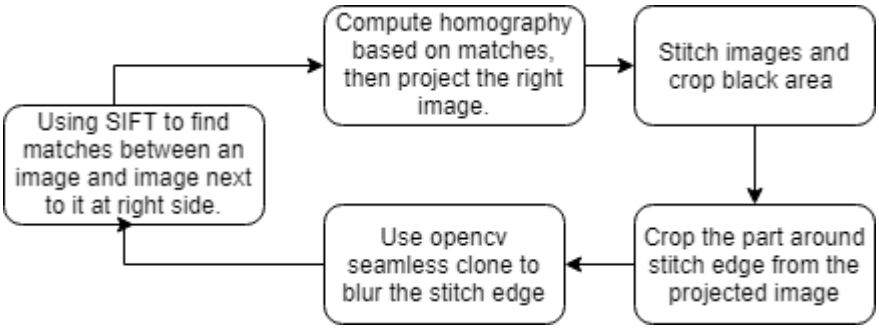
- 200 face images for each participant

METHODS

The main process is divided into several steps.

1. Panorama constructing
2. Distance computing
3. Training set gathering
4. Name identifying

PANORAMA



The above flow chart is self implemented panorama stiching algorithm

a. **Using SIFT to detect matches**
Scale-Invariant Feature Transform is a feature detection algorithm that we use to detect and describe local features in images to find matches between several images. Keypoints recognized by SIFT of an object are extracted from reference images, then they are used to recognize the object in new images by computing the Euclidean distance of their feature vectors.

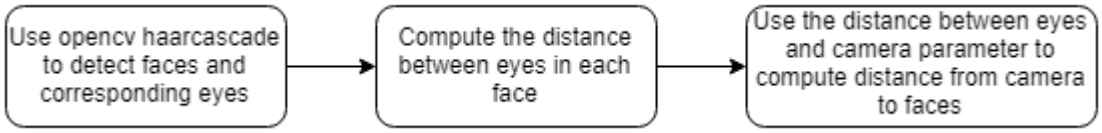
b. **Compute the Homography based on matches**
Matches gathered from previous step are filtered by applying a threshold to get quality matches that we can use to compute homogaphy matrix of the projected image. Computing Homography is necessary since in the image shooting process, the camera is rotating instead of moving in an axis, which means the adjacent image need to be projected to the same plane as previous image in order to stitch them together.

c. **Crop the black area to have better effect and save time for future computing**

d. **Blend the stitching result using the crop part of image around the stitch edge**
Repeating the above 4 process for all right side images to get right side stitched result. Then flip the stitched result and all left side images since out algorithm stitch images from left to right. By applying 4 above steps to them, we can get a flipped full panorama. Finally flip the image to get the final result.

```python
def stitch():
    """
    Read 9 images and stitch them to one panorama image
    :return: None, write generated image to disk
    """
    img1 = cv2.imread("landscape_1.jpg")
    img2 = cv2.imread("landscape_2.jpg")
    img3 = cv2.imread("landscape_3.jpg")
    img4 = cv2.imread("landscape_4.jpg")
    img5 = cv2.imread("landscape_5.jpg")
    img6 = cv2.imread("landscape_6.jpg")
    img7 = cv2.imread("landscape_7.jpg")
    img8 = cv2.imread("landscape_8.jpg")
    img9 = cv2.imread("landscape_9.jpg")
    out1 = img5
    for i in [img6, img7, img8, img9]:
        out1 = basic_wrap(out1, i)
    plt_show(out1)
    cv2.imwrite('rightside.jpg', out1)
    list2 = [np.flip(img4, axis=1), np.flip(img3, axis=1), np.flip(img2, axis=1), np.flip(img1, axis=1)]
    out2 = np.flip(out1, axis=1)
    for i in range(4):
        out2 = basic_wrap(out2, list2[i])
    out2 = np.flip(out2, axis=1)
    plt_show(out2)
    cv2.imwrite("result.jpg", out2)


def basic_wrap(B, A):
    """
    Stitch two adjacent images to one based on matching points
    :param B: One image
    :param A: Image right to image B
    :return: Generated image data
    """
    sift = cv2.xfeatures2d.SIFT_create()
    key1, desc1 = sift.detectAndCompute(A, None)
    key2, desc2 = sift.detectAndCompute(B, None)

    bf = cv2.BFMatcher()
    matches = bf.knnMatch(desc1, desc2, k=2)

    good = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good.append(m)

    src = np.array([key1[m.queryIdx].pt for m in good])
    dest = np.array([key2[m.trainIdx].pt for m in good])

    src = (src.reshape(-1, 1, 2)).astype(np.float)
    dest = (dest.reshape(-1, 1, 2)).astype(np.float)

    M, mask = cv2.findHomography(src, dest, cv2.RANSAC, 5.0)
    img_out = cv2.warpPerspective(A, M, (B.shape[1] + 700, B.shape[0] + 20))
    img_out[0: B.shape[0], 0: B.shape[1]] = B
    # plt_show(img_out)
    trim = np.where(~img_out.any(axis=0))[0]
    for i in range(len(trim)):
        try:
            img_out = np.delete(img_out, np.min(trim) - 10, axis=1)
        except:
            break
    # plt_show(img_out)
    origin = cv2.warpPerspective(A, M, (B.shape[1] + 700, B.shape[0] + 20))
    # plt_show(origin)
    img_paste = origin[0: B.shape[0], B.shape[1] - 100: B.shape[1] + 100]
    img_paste_crop = img_paste[10:, :]
    # plt_show(img_paste_crop)
    print(img_paste.shape, img_paste_crop.shape)
    mask = 255 * np.ones(img_paste_crop.shape, img_paste_crop.dtype)
    h, w = int(B.shape[1]), int(B.shape[0]/2)
    center = (h, w+5)
    img_new = cv2.seamlessClone(img_paste_crop, img_out, mask=mask, p=center, flags=cv2.NORMAL_CLONE, blend=None)
    # plt_show(img_new)
    cv2.imwrite('temp.jpg', img_new)
    return img_new
```

DISTANCE COMPUTING



a. Locate faces and corresponding eyes using opencv haarcascade xml file.
b. Compute distance between eyes in each faces.
c. Use the distance between eyes and camera parameter to compute distance from camera to faces

$$\text{Distance to object}(mm) = \frac{f(mm) \times \text{real height}(mm) \times \text{image height}(pixels)}{\text{object height}(pixels) \times \text{sensor height}(mm)}$$

The camera and lens we use to capture images are Canon EOS 80d with 16mm lens. Thereby we can get focal length f = 16 mm, original image size 4000*6000 pixel and sensor size 22.5 x 15 mm. Image size is reduced to 667*1000 to increase compute speed.

Average of distances between eyes is used as real height in the formula.

IPD values (mm) from 2012 Army Survey

Gender	Sample size	Mean	Standard deviation	Minimum	Maximum	Percentile				
						1st	5th	50th	95th	99th
Female	1986	61.7	3.6	51.0	74.5	53.5	55.5	62.0	67.5	70.5
Male	4082	64.0	3.4	53.0	77.0	56.0	58.5	64.0	70.0	72.5

Then we have all values we need to compute the distance between faces and camera.

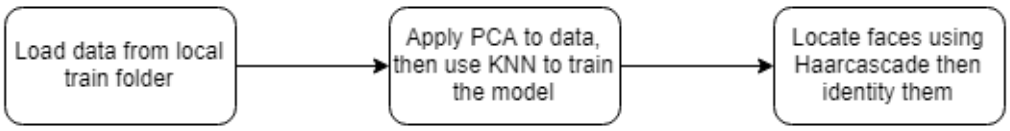
```
f = 16
real_height = 62
image_height = 1000
sensor_height = 22.5


def compute_distance(d):
    """
    computer distance from people in the image to camera
    :param d: Integer, distances between eyes of people in the image
    :return : Float, calculated distance from people in the image to lens
    """
    return f * real_height * image_height / (d * sensor_height)


def get_eye_distance(img):
    """
    Read image, detect faces in the image and use compute_distance function
    to calculate distances from people in the image to camera
    :param img: Image
    :return: None, show calculated distances in the image
    """
    img = cv2.imread(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.5, 4, minSize=(100, 100))
    print(faces)
    color = (255, 0, 0)
    font = cv2.FONT_HERSHEY_SIMPLEX
    for x, y, w, h in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        face_gray = gray[y:y + h, x:x + w]
        face_color = img[y:y + h, x:x + w]
        eyes = eye_cascade.detectMultiScale(face_gray)
        maybe_eyes = []
        for ex, ey, ew, eh in eyes:
            if ey < h/2:
                maybe_eyes.append((ex, ey, ew, eh))
        print(maybe_eyes)
        real_eyes = []
        for i in range(len(maybe_eyes)):
            for j in range(len(maybe_eyes)):
                if i != j:
                    if abs(maybe_eyes[i][0] - maybe_eyes[j][0]) > 0.01 * w and abs(maybe_eyes[i][1] - maybe_eyes[1][0]) < 0.1 * h:
                        real_eyes = [(maybe_eyes[i], maybe_eyes[j])]
        if len(real_eyes) < 2:
            real_eyes = maybe_eyes[:2]
        eye_dist = abs(real_eyes[0][0] - real_eyes[1][0])
        dist = compute_distance(eye_dist)
        for ex, ey, ew, eh in real_eyes:
            cv2.rectangle(face_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)
        cv2.putText(img, 'distance: %dcm' % (dist/10), (x + 10, y - 20), font, 1, color, 3)
    plt_show(img)
    return None
```

COLLECT TRAINING IMAGE

a. Use opencv videocapture to get frames from the video
b. Locate faces using haarcascade
c. Crop faces and save them as train images in train folder
Training images should include multiple angles and multiple conditions.

```python
def capture_trainset(video_name, window_name, pic_num, path_name):
    """
    Capture face images from the video as training data for face recognition model
    :param video_name: path and name of the video file
    :param window_name: cv2 window name to open video
    :param pic_num: number of pictures we want to capture from the video
    :param path_name: path to the folder to store captured images
    :return: None, save images to disk
    """
    cv2.namedWindow(window_name)
    cap = cv2.VideoCapture(video_name)
    classfier = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
    color = (255, 0, 0)
    font = cv2.FONT_HERSHEY_SIMPLEX
    num = 0
    while cap.isOpened():
        status, frame = cap.read()
        if not status:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # use classifier to detect faces in the frame
        faces = classfier.detectMultiScale(gray, 1.3, 2, minSize=(300, 300))
        if len(faces) != 0:
            for x, y, w, h in faces:
                img_name = "%s/%s%d.jpg" % (path_name, video_name[0], num)
                print(img_name)
                image = frame[y - 10: y + h + 10, x - 10: x + w + 10]
                cv2.imwrite(img_name, image)
                num += 1

                # when capture enough pictures, end the program
                if num > pic_num:
                    break

                cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), color, 2)
                cv2.putText(frame, 'saved:%d/%d' % (num, pic_num), (x + 10, y + 10), font, 1, color, 4)
        if num > pic_num:
            break

        cv2.imshow(window_name, frame)
        c = cv2.waitKey(25)
        if c & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    capture_trainset("r.mp4", "get face", 499, "./train")
```

TRAIN KNN MODEL TO IDENTIFY NAME

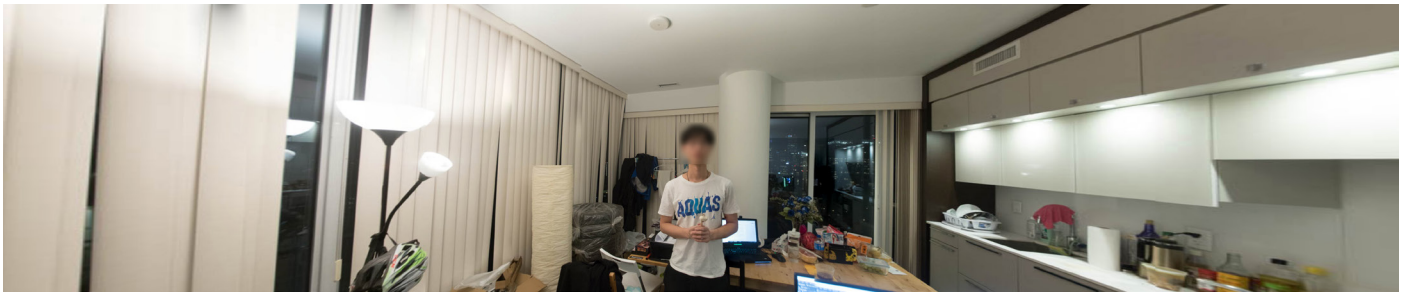


a. Load training images captures from previous step
b. Using PCA to speed up the algorithm, then use KNN to train the model.
c. In K-Nearest Neighbors, nearest k training instances to the target instance is recorded.
d. Locate faces using Haarcascade, then use the trained model to predict the class.

```
def load_data():
    """
    Load all training image data from train folder and transform them into numpy array
    :return: Two numpy array (Training data and label)
    """
    data, label = [], []
    num = 270
    path = "./train/"
    pid = ['r','c']
    for j in range(len(pid)):
        for number in range(num):
            path_full = path + pid[j] + str(number) +'.jpg'
            image = Image.open(path_full).convert('L')
            image = image.resize((100, 100), Image.ANTIALIAS)
            img = np.reshape(image, (1, 100*100))
            data.extend(img)
        label.extend(j * np.ones(num, dtype=np.int))
    data = np.reshape(data, (-1, 100*100))
    return np.matrix(data), np.matrix(label).T


def knn(neighbor, traindata, trainlabel, testdata):
    """
    Fit and predict the KNN model
    :param neighbor: Integer, number of neighbor
    :param traindata: Numpy array, training image data
    :param trainlabel: Numpy array, training label
    :param testdata: Numpy array, image data to be predicted
    :return: Integer, predicted result
    """
    neigh = KNeighborsClassifier(n_neighbors=neighbor)
    neigh.fit(traindata, trainlabel)
    result = neigh.predict(testdata)
    return result


if __name__ == '__main__':
    # load data from train folder and pca feature extraction
    data, label = load_data()
    pca = PCA(n_components=0.9, whiten=True)
    pca_data = pca.fit_transform(data)
    color = (255, 0, 0)
    cap = cv2.VideoCapture("test.mp4")
    while cap.isOpened():
        status, frame = cap.read()
        if not status:
            break
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        classifier = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")

        # detect faces in the frame
        faces = classifier.detectMultiScale(frame_gray, 1.3, 5, minSize=(150, 150))
        if len(faces) > 0:
            for x, y, w, h in faces:
                # crop the face
                image = frame_gray[y - 10: y + h + 10, x - 10: x + w + 10]

                # resize the crop part then use knn to recognize
                image = cv2.resize(image, (100, 100))
                img_test = np.reshape(image, (1, 100 * 100))
                pca_test = pca.transform(img_test)
                result = knn(5, pca_data, label, pca_test)
                faceID = result[0]
                print(faceID)

                # find corresponding faces and print faces
                if faceID == 0:
                    cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), color, thickness=2)
                    cv2.putText(frame,'Robert', (x + 10, y + 20), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
                elif faceID == 1:
                    cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), color, thickness=2)
                    cv2.putText(frame, 'Cheney', (x + 10, y + 20), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
                else:
                    cv2.rectangle(frame, (x - 10, y - 10), (x + w + 10, y + h + 10), color, thickness=2)
                    cv2.putText(frame, 'Unknown', (x + 10, y + 10), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)

        cv2.imshow("detect", frame)
        k = cv2.waitKey(25)
        if k & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

RESULTS AND DISCUSSIONS

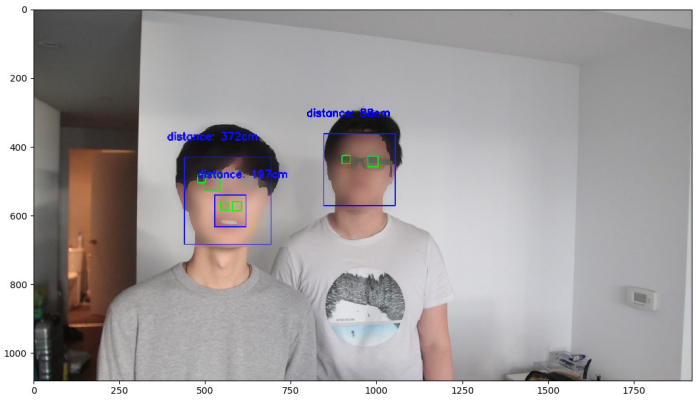
RESULTS FOR PANORAMA STICHING



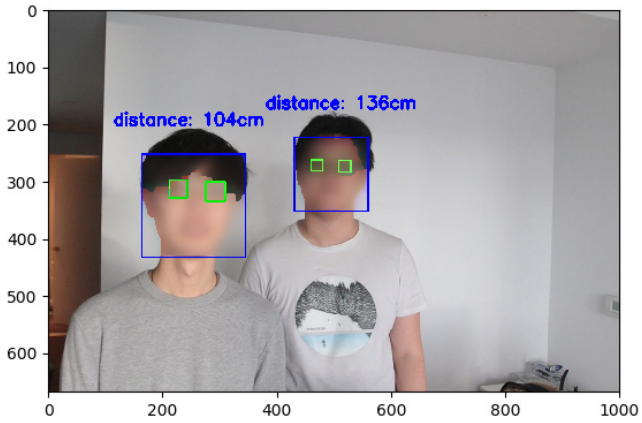
The algorithm works as expected. There is a small mismatch at bottom part (monitor) since I use a wide angle lens to capture the picture which cause the distortion of original images. To eliminate this error, we can use a 50mm lens instead of 16mm to capture original images with accurate shape.

In addition, only center part of the panorama picture we created can be used in later steps to recognize face since sides part of the image are projected image. To fix this distortion , we can use other panorama stitching method like spherical or cylindrical instead of perspective (cv2.Stitcher_ create).

DISTANCE COMPUTING



Result before tuning parameter

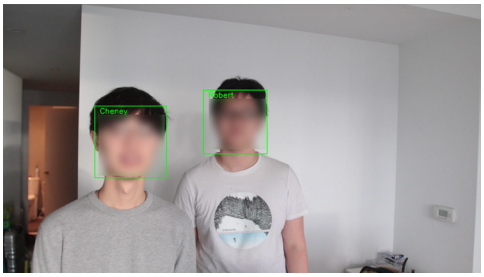


Final Result

At first, the algorithm cannot locate faces and eyes correct. For instance, nostrils, mouths and places near hairs are always detected as eyes. By setting appropriate min size when classify faces, we can recognize faces correct. Then filter all eyes detected in bottom half faces to eliminate wrong detections at nostrils and mouths. Finally pick two detections with most similar size and height as eyes location to compute the distance.

This algorithm will not work if eyes cannot be detected in the face. Sometimes this happens on subjects wearing glass, or they only show side faces in the frame.

NAME IDENTIFY



Same as previous step, nose and mouth may be detected as faces without setting min size for face detection. After testing, k=5 for KNN will produce the most accurate result. The error rate can be further reduced by creating better training set. (training sets includes more angles, more light conditions and more facial expressions)

MAIN CHALLENGES

The main challenges of this project are parameter tuning and parameter picking in distance computing step. In order to get accurate location of faces and eyes, parameter such as scaleFactor and MinSize in CascadeClassifier need to be tuned. For different pictures, those parameters need to be adjusted every time since ratios of face size compared with image size are different.

Pupillary distance (distance between eyes) is used to compute the distance between faces and cameras. We have tried other distance that can be identified on the face such as single eye size and face length, but both of them did not perform well. For eye size, the result is incorrect since eye size of different people have large standard deviation, thereby mean of eye size cannot be used as real height in the formula to compute distance. For face length, the result is incorrect since face lengths detected in the image are usually not precise. The face detector in haarcascade only consider faces under hairline as faces, thereby it cannot be used as object height in the formula to compute distance.

FURTURE WORK

The overall quality of this project can be enhanced if panorama and training set capturing process are improved. For panorama stitching, using spherical or cylindrical transform can produce a panorama picture with less distortion at sides. For training set capturing, more angles of faces need to be captured (even side faces) to improve the accuracy of name identify process.

LEAGUE OF LEGENDS
WINRATE PREDICTION

GOALS

The goal of this project is to build several machine learning models to predict the winrate of a League of Legends game before the game start using team composition and during game using in-game data.
The project can be separated into 3 parts.

1. Machine learning model that predict game winrate based on purely team composition
2. Machine learning model that extract real time game data from game screen
3. Machine learning model that predict game winrate using team composition and realtime in-game data

TOOLS

- Pandas
- OpenCV
- Scikit-lean

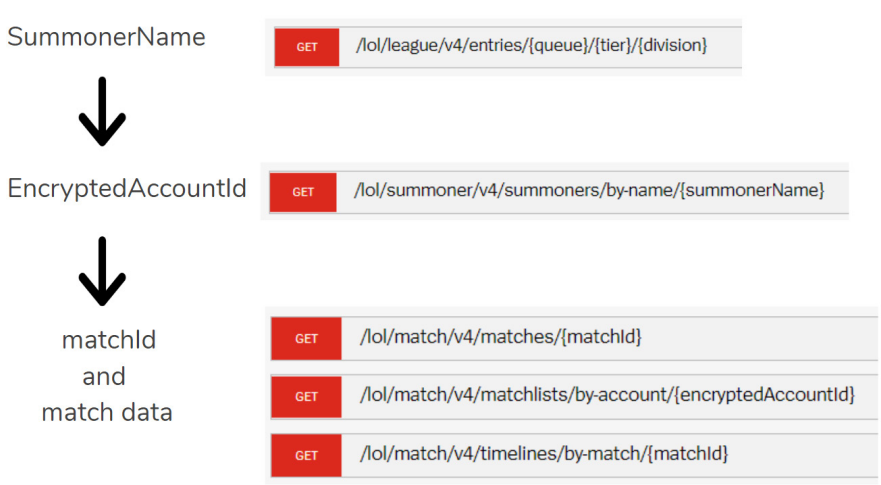
DATA

- 400,000 game matches data including team composition, in-game data, and result
- digits images for digit recognition

DATA GATHERING AND CLEANING

League of Legends is a popular 5v5 online MOBA game with over 100 million players all over the world. Due to the popularity, they have a public developer portal to collect data through api.

1. Collect summoner name in Platinum above (top 15% players in the server) since data of low tier players is not representative
2. Use collected summonernames to get their encrypted account ids
3. Use encrypted account id to get list of match id



```python
def get_sum_from_league(division, tier, pagenum, api_key):
    """
    return pandas datafram contain summoner info
    :param division: str
    :param tier: str
    :param pagenum: int
    :param api_key: str
    :return: dataframe
    """
    summoners = {'summonerId': [], 'summonerName': []}
    for p in range(1, pagenum):
        url = f'https://na1.api.riotgames.com/lol/league/v4/entries/RANKED_SOLO_5x5/{division}/{tier}' \
              f'?page={p}&api_key={api_key}'
        response = requests.get(url)
        time.sleep(1)
        data = response.json()
        for i in data:
            summoners['summonerId'].append(i['summonerId'])
            summoners['summonerName'].append(i['summonerName'])
    return pd.DataFrame(summoners)


def get_all_sum_info(summonerName, api_key):
    """
    get detail summoner info based on given summoner name
    :param summonerName: str
    :param api_key: str
    :return: pandas df contain summonerId, summonerName, accountId, puuid
    """
    summoners = {'summonerId': [], 'summonerName': [], 'accountId': [], 'puuid': []}
    url = f'https://na1.api.riotgames.com/lol/summoner/v4/summoners/by-name/{summonerName}?api_key={api_key}'
    response = requests.get(url)
    data = response.json()
    print(data)
    try:
        summoners['summonerId'].append(data['id'])
        summoners['summonerName'].append(data['name'])
        summoners['accountId'].append(data['accountId'])
        summoners['puuid'].append(data['puuid'])
        return pd.DataFrame(summoners)
    except:
        return pd.DataFrame()
```

example code for using the api

After collecting raw data from api, drop rows with null values due to network/server issue.

For model 1 training data:

Extract champion selection columns and result columns from the original table after cleaning

For model 1 test data:

Champion selection data of ongoing match can be collected from api

For model 2 training and test data:

Image of digits and in-game event ui. This will be mentioned in detail later in the machine learning implementing section

For model 3 training data:

Extract champion selection columns, relative match data columns, and result columns from the original table after cleaning

For model 3 test data:

In-game data produced by using model 2 during game

Since we want to use team composition as a feature in our machine learning models, first we encode the champion data to convert categorical columns to numerical. There are 148 champions now in the game in total. Therefore, after encoding, there are 148 columns. The value of a champion column is 1 if team 1 selects this champion, 0 if no team selects this champion, and -1 if team 2 select this champion.

```
import pandas as pd

df = pd.read_csv('full_matchdata.csv', index_col=0)
df_mtl = pd.read_csv('full_matchtimeline.csv', index_col=0)
df_match_timeline = df_mtl[['gameId', 'game_time']]
df_col = list(df.columns)

df_t1_champ = df[['team1_p1_championId', 'team1_p2_championId', 'team1_p3_championId', 'team1_p4_championId',
                  'team1_p5_championId']]
df_t2_champ = df[['team2_p1_championId', 'team2_p2_championId', 'team2_p3_championId', 'team2_p4_championId',
                  'team2_p5_championId']]
df_t1_win = df['team1_win']

df_champ_keys = pd.read_csv('champ_keys.csv')
# df_champ_keys_new = df_champ_keys[['Champ_Name', 'Champ_Key']]

champ_names = list(df_champ_keys['Champ_Name'])
champ_keys = list(df_champ_keys['Champ_Key'])
champ_index = list()

'''create a dict of each matches
champions as column names
1 stands for team1 having this champion
-1 stands for team2 having this champion
0 stands for both or neither team have this champion'''
champ_picks = {}
for i in range(len(df_t1_win)):
    lst = [0] * len(champ_names)
    for key1 in df_t1_champ.loc[i]:
        index1 = champ_keys.index(key1)
        lst[index1] += 1
    for key2 in df_t2_champ.loc[i]:
        index2 = champ_keys.index(key2)
        lst[index2] -= 1
    champ_picks[i] = lst

df_team1_win = df[['gameId', 'team1_win']]
df_c = pd.DataFrame.from_dict(champ_picks, orient='index', columns=champ_names)
df_champ = pd.concat([df_c, df_team1_win], axis=1)
df_champ_select = pd.merge(df_champ, df_match_timeline, on='gameId')
df_champ_select.to_csv('champ_select.csv')

df_win = df[['gameId', 'team1_win', 'team2_win']]
df_win.to_csv('game_win.csv')

df_objectives = df[['gameId', 'team1_win', 'team2_win', 'team1_baronKills', 'team1_riftKills', 'team1_inhibitorKills',
                    'team1_towerKills', 'team1_dragonKills', 'team2_baronKills', 'team2_riftKills',
                    'team2_inhibitorKills', 'team2_towerKills', 'team2_dragonKills', 'team1_firstDragon',
                    'team1_firstInhibitor', 'team1_firstRiftHerald', 'team1_firstBaron', 'team1_firstBlood',
                    'team1_firstTower', 'team2_firstDragon', 'team2_firstInhibitor', 'team2_firstRiftHerald',
                    'team2_firstBaron', 'team2_firstBlood', 'team2_firstTower']]
df_objectives.to_csv('game_objectives.csv')
```

MODEL IMPLEMENTING

Since the goal is predicting winrate instead of predicting win/lose, I choose to use the prediction probability as the result instead of using the classification result.
For these two prediction models, I decided to use XGBoost.

TEAM COMPOSITION MODEL

Feature Engineering

Apart from encoded champion selection, I also add a general team composition that counts types of champion in each team (e.x. how many tanks/assasins/mages/ etc in each team) to enhance the model performance.

Training Model

After feature engineering, I just split train test data and standardized data to fit the XGBoost Classifier. Since the data size is pretty big, cross validation is not required.

The model accuracy of model with default hyperparameter is only around 0.6, after tuning the accuracy increases to 0.75

INGAME DATA MODEL

Feature Engineering

Raw ingame data has massive amounts of columns. Based on my game experience, I only select features that might affect the game result. Then I check correlations between those selected features and win/lose to further select appropriate features. After filtering, I choose to use team objectives data (dragons, barons, towers, etc) and individual player data (creep score, level, and Gold) as input features.

Game time is also an important feature since some team compositions are strong at early game and some team compositions are strong at late game team fights, but time itself should not affect prediction result without other time-based features. My approach to handle time is building separate models for each 10 min intervals. The average game time is around 30 min. Therefore, I have 4 models for time intervals 0-10 min, 10-20 min, 20-30 min, and 30+min. Since the time data now is discrete instead of continuous, in order to use real time in-game data like current gold, I decide to use gold per min as a feature instead of using the total amount of gold.

Training Model

The process is the same as the previous model.

The model accuracy of this model with default hyperparameter for 0-10 min is only around 0.55 since early game stats do not have a significant impact on game results. For other models, the accuracy after parameter tuning is around 0.8.

IMAGE RECOGNITION MODEL

In order to use the model that use in-game data to predict, I need to extract data from game screen during game. I use opencv to capture screen for extracting data. The screen will be captured every 2 seconds to reduce computation costs.

Things I want to collect from the screen:
Objectives: Dragons, Rift Heralds, Dragons, Towers, Inhibitors
Players: Creep Score, kill/death/assist

Example game screen with scoreboard

```python
def train_digits(img):
    """
    create training data by manually input number label, save training data to file
    :return: None
    """
    im = cv2.imread(img)
    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.THRESH_BINARY_INV, 1, 3, 3)

    # finding Contours

    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    samples = np.empty((0, 100))
    responses = []
    keys = [i for i in range(47, 58)]

    for cnt in contours:
        if 5 < cv2.contourArea(cnt) < 100:
            [x, y, w, h] = cv2.boundingRect(cnt)
            print(w, h)
            if 10 < h and 6 <= w:
                temp = im.copy()
                cv2.rectangle(temp, (x, y), (x + w, y + h), (0, 0, 255), 1)
                roi = thresh[y:y + h, x:x + w]
                roismall = cv2.resize(roi, (10, 10))
                cv2.imshow('norm', temp)
                key = cv2.waitKey(0)

                if key == 27:  # (escape to quit)
                    sys.exit()
                elif key in keys:
                    if key == 47:
                        responses.append(11)
                    else:
                        responses.append(int(chr(key)))
                    sample = roismall.reshape((1, 100))
                    samples = np.append(samples, sample, 0)

    responses = np.array(responses, np.float32)
    responses = responses.reshape((responses.size, 1))

    np.savetxt('generalsamples1.data', samples)
    np.savetxt('generalresponses1.data', responses)
```

Player data can be extracted from the scoreboard, we just need a digit recognition model. I capture scoreboards several times to get all digits, then use OpenCV findContours function to create training image for all digits.

Digits cropped from sample scoreboard using function train_digits at left, then save them as nx1 dimension numpy array for training purpose.

```
# get player cs and kda
keyboard.wait('tab')
prev_team1_p1_cs, prev_team1_p1_kills, prev_team1_p1_deaths, prev_team1_p1_assists = team1_p1_cs, team1_p1_kills, team1_p1_deaths, team1_p1_assists
prev_team1_p2_cs, prev_team1_p2_kills, prev_team1_p2_deaths, prev_team1_p2_assists = team1_p2_cs, team1_p2_kills, team1_p2_deaths, team1_p2_assists
prev_team1_p3_cs, prev_team1_p3_kills, prev_team1_p3_deaths, prev_team1_p3_assists = team1_p3_cs, team1_p3_kills, team1_p3_deaths, team1_p3_assists
prev_team1_p4_cs, prev_team1_p4_kills, prev_team1_p4_deaths, prev_team1_p4_assists = team1_p4_cs, team1_p4_kills, team1_p4_deaths, team1_p4_assists
prev_team1_p5_cs, prev_team1_p5_kills, prev_team1_p5_deaths, prev_team1_p5_assists = team1_p5_cs, team1_p5_kills, team1_p5_deaths, team1_p5_assists

prev_team2_p1_cs, prev_team2_p1_kills, prev_team2_p1_deaths, prev_team2_p1_assists = team2_p1_cs, team2_p1_kills, team2_p1_deaths, team2_p1_assists
prev_team2_p2_cs, prev_team2_p2_kills, prev_team2_p2_deaths, prev_team2_p2_assists = team2_p2_cs, team2_p2_kills, team2_p2_deaths, team2_p2_assists
prev_team2_p3_cs, prev_team2_p3_kills, prev_team2_p3_deaths, prev_team2_p3_assists = team2_p3_cs, team2_p3_kills, team2_p3_deaths, team2_p3_assists
prev_team2_p4_cs, prev_team2_p4_kills, prev_team2_p4_deaths, prev_team2_p4_assists = team2_p4_cs, team2_p4_kills, team2_p4_deaths, team2_p4_assists
prev_team2_p5_cs, prev_team2_p5_kills, prev_team2_p5_deaths, prev_team2_p5_assists = team2_p5_cs, team2_p5_kills, team2_p5_deaths, team2_p5_assists

team1_p1_cs = get_cs(team1_p1_cs_area)
team1_p1_kills, team1_p1_deaths, team1_p1_assists = get_kda(team1_p1_kda_area)
team1_p2_cs = get_cs(team1_p2_cs_area)
team1_p2_kills, team1_p2_deaths, team1_p1_assists = get_kda(team1_p2_kda_area)
team1_p3_cs = get_cs(team1_p3_cs_area)
team1_p3_kills, team1_p3_deaths, team1_p1_assists = get_kda(team1_p3_kda_area)
team1_p4_cs = get_cs(team1_p4_cs_area)
team1_p4_kills, team1_p4_deaths, team1_p1_assists = get_kda(team1_p4_kda_area)
team1_p5_cs = get_cs(team1_p5_cs_area)
team1_p5_kills, team1_p5_deaths, team1_p1_assists = get_kda(team1_p5_kda_area)
```

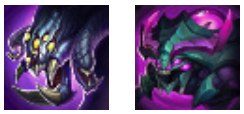
For player data (creep score, kill/death/assist), I specify the position to find them on the captured screen, then use the model to get digits from that area. Since the scoreboard is only shown when a player press key tab in game, the model will only run when a player press tab each time in get_cs and get_kda functions. In kda function, k/d/a is separated by a slash. In cs function, the score is computed using digits detected and their positions.

```
def get_kda(scene):
    """
    return recognized minions killed
    :param scene:
    :return: int
    """
    # fit model using training data
    samples = np.loadtxt('generalsamples.data', np.float32)
    responses = np.loadtxt('generalresponses.data', np.float32)
    responses = responses.reshape((responses.size, 1))
    model = cv2.ml.KNearest_create()
    model.train(samples, cv2.ml.ROW_SAMPLE, responses)
    # predict digit
    gray = cv2.cvtColor(scene, cv2.COLOR_RGB2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 3, 4)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    digits = []
    position = []
    appeared = {}
    for cnt in contours:
        if 5 < cv2.contourArea(cnt) < 150:
            [x, y, w, h] = cv2.boundingRect(cnt)
            if h > 8:
                roi = thresh[y:y + h, x:x + w]
                roismall = cv2.resize(roi, (10, 10))
                roismall = roismall.reshape((1, 100))
                roismall = np.float32(roismall)
                retval, results, neigh_resp, dists = model.findNearest(roismall, k=1)
                if x-1 not in appeared and x+1 not in appeared and x not in appeared:
                    appeared[x] = dists[0][0]
                    if dists[0][0] < 900000 or ((results[0][0] == 1 or results[0][0] == 11) and dists[0][0] < 1500000):
                        digit = int((results[0][0]))
                        digits.append(digit)
                        position.append(x)
                elif x-1 in appeared:
                    if dists[0][0] < appeared[x-1]:
                        digit = int(results[0][0])
                        digits.append(digit)
                        position.append(x)
                elif x+1 in appeared:
                    if dists[0][0] < appeared[x+1]:
                        digit = int(results[0][0])
                        digits.append(digit)
                        position.append(x)
                elif x in appeared:
                    if dists[0][0] < appeared[x]:
                        digit = int(results[0][0])
                        digits.append(digit)
                        position.append(x)
    # print(digits)
    if len(digits) != 0:
        position, digits = zip(*sorted(zip(position, digits)))
        slash_index = [i for i, d in enumerate(digits) if d == 11]
        if len(slash_index) != 0 and len(slash_index) == 2 and slash_index[1] + 1 <= len(digits):
            # print(slash_index)
            # print(digits)
            kills_lst = digits[:slash_index[0]]
            deaths_lst = digits[slash_index[0] + 1: slash_index[1]]
            assists_lst = digits[slash_index[1] + 1:]
            kills = concat_number_from_lst(kills_lst)
            deaths = concat_number_from_lst(deaths_lst)
            assists = concat_number_from_lst(assists_lst)
            return kills, deaths, assists
    return 0, 0, 0
```

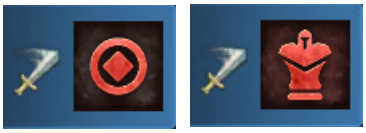
Team data of objectives can be extracted from the most right part of the screen. When an event happens, a notification will pop up. Blue represents team action, red represents enemy action.

Event icon

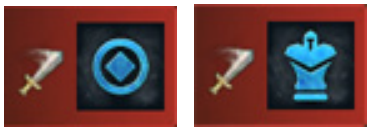
Dragon icon



Baron/Herald icon



team destroy eneymy inhibitors/towers



team inhibitors/towers are destroyed

```python
def event_template_match(scene, template, th, sw, debug=False):
    res = cv2.matchTemplate(scene, template, cv2.TM_CCOEFF_NORMED)
    minmax = cv2.minMaxLoc(res)
    _, max_val, _, max_loc = minmax
    corner1 = max_loc
    corner2 = (max_loc[0] + template[1], max_loc[1] + template[0])
    # cv2.rectangle(scene, corner1, corner2, (255, 0, 0), 3)
    # cv2.imshow("output", scene)
    prev_sw = sw
    if debug:
        print(f'prob: {max_val}, sw: {sw}, prev_sw: {prev_sw}')
    if max_val > th:
        sw = 1
    else:
        sw = 0

    if sw == 0 and prev_sw == 1:
        print('event happened')
        return 1, 0, max_val
    return 0, sw, max_val
```

By using OpenCV matchTemplate function, we can try to find these event icons on screen. matchTemplate function will return similarity between the area in scene and template. By setting threshold, these icons can be detected can be recognized perfectly.
Unlike player data, an event happens all the time. So this function is running constantly without any trigger condition

At this point, we have collected all data required to use the in-game data model. The model will run when there is a change in data to generate a predicted winrate.

SUMMARY & FUTURE IMPROVEMENT

- Overall, after testing the algorithm in several matches, it gives reasonably predicted winrate
- For future improvement, we can try to find out better ways to use time-series data. And for the model using team composition, in League of Legends, there are actually some champion combos that have special interaction, but our model only considers each champion as an individual, so combo will not have an effect on winrate as it should be.

VANCOUVER HOUSE
PRICE ANALYSIS AND PREDICTION

GOALS

- Use online house information to analyze the trend of house market in Vancouver
- Implement a regression model to predict the price of a house
- Implement a cluster model to separate houses into groups
- Build online dashboard for use to see the analysis and use the model

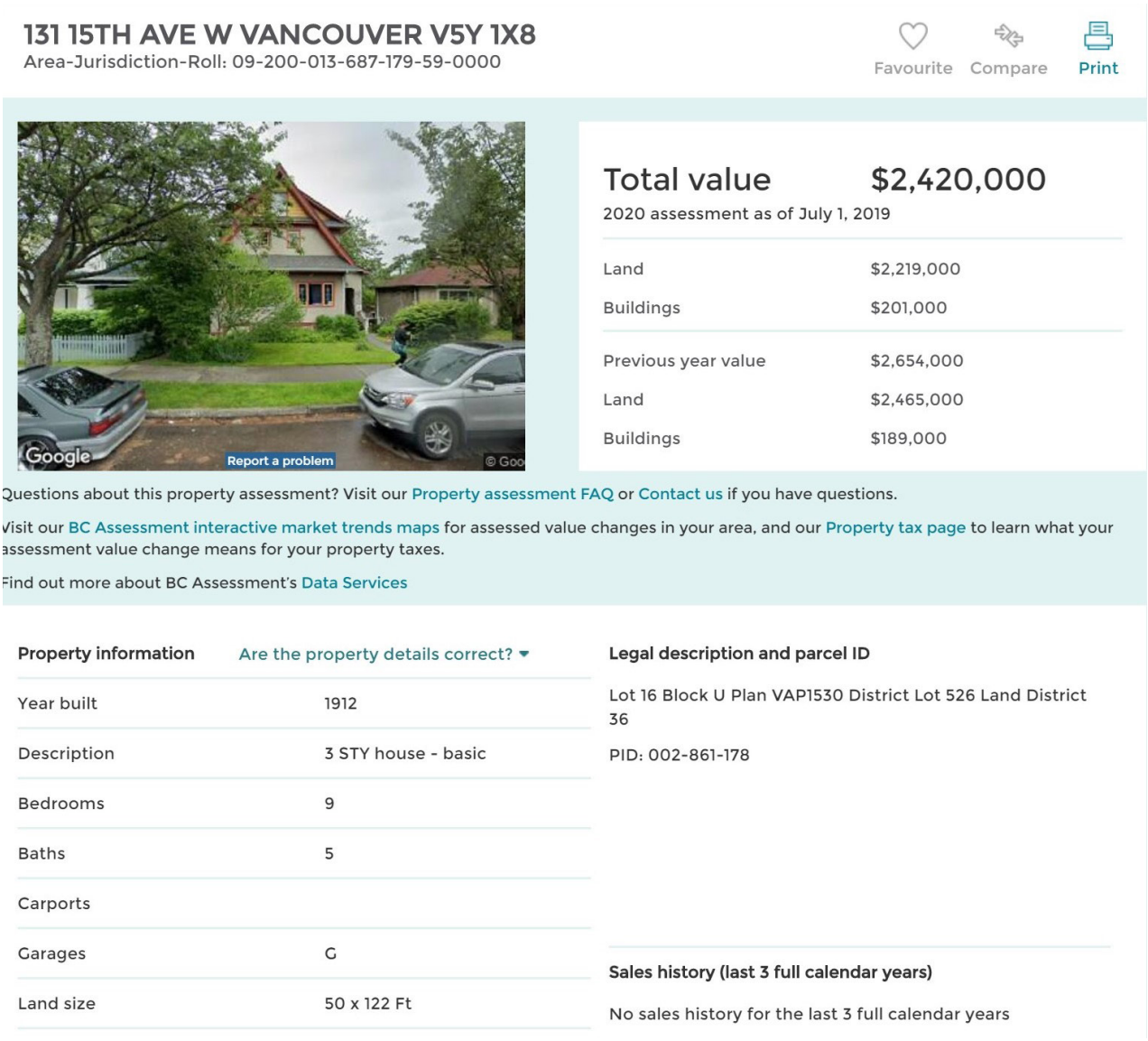
TOOLS

- Web scrapping: Beautiful soup, Selenium
- Data cleaning and pre-processing: Pandas
- Machine learning: Scikit-learn
- Visualization: Plotly Dash, Folium

DATA

- Scraped house data includes address, price, previous year price, area, number of bedrooms, number of bathrooms, garage, and carport.

DATA GATHERING AND CLEANING



131 15TH AVE W VANCOUVER V5Y 1X8
Area-Jurisdiction-Roll: 09-200-013-687-179-59-0000

Favourite Compare Print

Total value $2,420,000
2020 assessment as of July 1, 2019

Land	$2,219,000
Buildings	$201,000
Previous year value	$2,654,000
Land	$2,465,000
Buildings	$189,000

Questions about this property assessment? Visit our Property assessment FAQ or Contact us if you have questions.

Visit our BC Assessment interactive market trends maps for assessed value changes in your area, and our Property tax page to learn what your assessment value change means for your property taxes.

Find out more about BC Assessment's Data Services

Property information — Are the property details correct?

Year built	1912
Description	3 STY house - basic
Bedrooms	9
Baths	5
Carports	
Garages	G
Land size	50 x 122 Ft

Legal description and parcel ID

Lot 16 Block U Plan VAP1530 District Lot 526 Land District 36

PID: 002-861-178

Sales history (last 3 full calendar years)

No sales history for the last 3 full calendar years

A example of house information on bcassessment.ca

- In order to use the website to search, we can either use the specific civic address of the building or use its parcel id. I went to Vancouver Open Data Portal to download a csv including all buildings' addresses and parcel id. After that, I used parcel id to get the building page in selenium, save the html page source and use Beautiful Soup to parse the html to get useful information including current value, previous year value, built year, area, number of bedrooms, number of bathrooms and garage.
- Since the website has an anti-scraping algorithm, I need to set a random wait timer from 5 to 10 seconds after each search. To speed up the process, I launched 4 AWS EC2 Instances to run the scraping code, then store the collected data to a S3 bucket.

Code in next page

```
pid_df = pd.read_csv('./buildings.csv', index_col=0)
pid_df = pid_df.dropna(subset=['SITE_ID'])

driver = webdriver.Chrome('./chromedriver.exe')
for index, row in pid_df.iterrows():
    # to not get blocked, quit webdriver after search for 10 results
    if index % 10 == 0:
        driver.quit()
        driver = webdriver.Chrome('./chromedriver.exe')
    pid = row['SITE_ID']
    full_address = row['full_add']
    driver.get("https://www.bcassessment.ca/")
    if pid.isdigit():
        select = Select(driver.find_element_by_id('ddlSearchType'))
        select.select_by_visible_text('PID')
        driver.find_element_by_id('txtPID').send_keys(pid)
        driver.find_element_by_id('btnSearch').send_keys(Keys.ENTER)
    else:
        select = Select(driver.find_element_by_id('ddlSearchType'))
        select.select_by_visible_text('Civic address')

        driver.find_element_by_id('rsbSearch').send_keys(full_address)
        element = WebDriverWait(driver, 10).until(
            EC.visibility_of_element_located((By.CSS_SELECTOR, ".ui-menu-item-wrapper"))
        )
        element.click()

    time.sleep(random.uniform(2, 8))
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'lxml')
    try:
        pid_df.loc[index, 'actualy_address'] = soup.find(id='mainaddresstitle').contents[0]
    except:
        pid_df.loc[index, 'actualy_address'] = np.nan
    try:
        pid_df.loc[index, 'total_value'] = soup.find(id='lblTotalAssessedValue').contents[0]
    except:
        pid_df.loc[index, 'total_value'] = np.nan
    try:
        pid_df.loc[index, 'prev_value'] = soup.find(id='lblPreviousAssessedValue').contents[0]
    except:
        pid_df.loc[index, 'prev_value'] = np.nan
    try:
        pid_df.loc[index, 'built_year'] = soup.find(id='lblYearBuilt').contents[0]
    except:
        pid_df.loc[index, 'built_year'] = np.nan
    try:
        pid_df.loc[index, 'bedroom'] = soup.find(id='lblBedrooms').contents[0]
    except:
        pid_df.loc[index, 'bedroom'] = np.nan
    try:
        pid_df.loc[index, 'bathroom'] = soup.find(id='lblBathRooms').contents[0]
    except:
        pid_df.loc[index, 'bathroom'] = np.nan
    try:
        pid_df.loc[index, 'garage'] = soup.find(id='lblGarages').contents[0]
    except:
        pid_df.loc[index, 'garage'] = np.nan
    try:
        pid_df.loc[index, 'carport'] = soup.find(id='lblCarPorts').contents[0]
    except:
        pid_df.loc[index, 'carport'] = np.nan
    try:
        pid_df.loc[index, 'area'] = soup.find(id='lblLandSize').contents[0]
    except:
        pid_df.loc[index, 'area'] = np.nan
    print(pid_df.loc[index])

pid_df.to_csv('scraped.csv', index=False)
```

DATA CLEANING

Due to network error, for some search, there is no result returned, so I have to drop those empty rows. For current value and previous value, parse the string and convert them to integer for future use.
For bedroom and bathroom, I fill null values with 0 since building with null bedroom and bathroom is commercial building or condo/apartment. For garage, in raw data 'G' means the house has a garage, so I just change 'G' to 1 and fill null with 0. Finally, for area, I split the string on space, if there is x in the string, multiply the dimension to get area, otherwise, just parse the area then convert to float.
After cleaning, subtract previous value from current value to get price change, divide price change by previous value to get price change rate, divide the current value by area to get the unit price.

```
df = pd.read_csv('final.csv', index_col=0)

# change int column to Int64 to handle missing value
for i in ['CIVIC_NUMBER', 'bathroom', 'bedroom']:
    df[i] = df[i].astype('Int64')

# drop rows with null price, area and built_year, then clean data
df = df.dropna(subset=['total_value', 'prev_value', 'area'])
df['total_value'] = df['total_value'].apply(lambda x: int(x.replace('$', '').replace(',', '')))
df['prev_value'] = df['prev_value'].apply(lambda x: int(x.replace('$', '').replace(',', '')))
df['garage'] = df['garage'].apply(lambda x: 1 if not pd.isnull(x) else 0)
df['carport'] = df['carport'].apply(lambda x: 1 if not pd.isnull(x) else 0)
df['built_year'] = df['built_year'].apply(lambda x:  int(x) if x != ' ' else np.nan)
df['bedroom'] = df['bedroom'].fillna(0).astype(int)
df['bathroom'] = df['bathroom'].fillna(0).astype(int)


def get_area(str_area):
    str_lst = str_area.split()
    if 'x' in str_lst:
        x_index = str_lst.index('x')
        return float(str_lst[x_index - 1]) * float(str_lst[x_index + 1])
    if 'Sq' in str_lst:
        sq_i = str_lst.index('Sq')
        return float(str_lst[sq_i - 1])


df['area'] = df['area'].apply(lambda x: get_area(x))
df['unit_price'] = df['total_value'] / df['area']
df['price_change'] = df['total_value'] - df['prev_value']
df['change_rate'] = df['price_change'] / df['prev_value']

df.to_csv('cleaned.csv', index=False)
```

Model Implementing and Visualization

MACHINE LEARNING

- Using Linear Regression to predict residential house prices based on Region, built year, area, number of bedrooms, bathrooms, and garages.
- Since the variance of values is big, I decide to train separate models from each individual region to improve the accuracy. Normalize the data since the value range difference of columns is big.

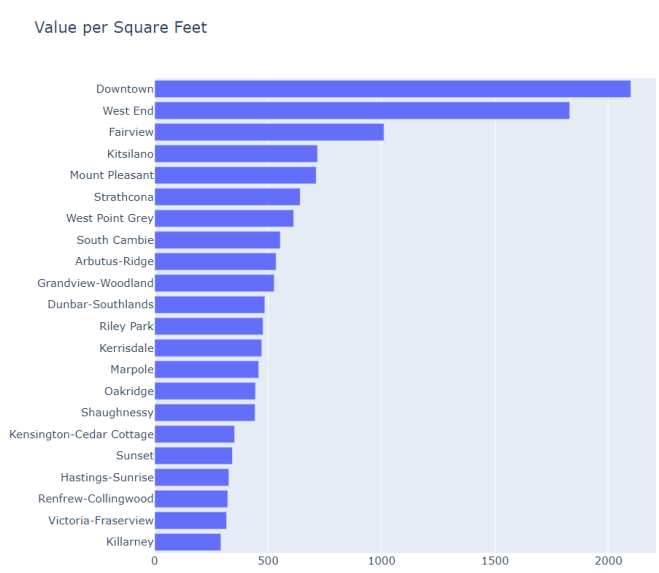
```
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from joblib import dump


regions = ['Kensington-Cedar Cottage', 'Renfrew-Collingwood', 'Sunset', 'Hastings-Sunrise', 'Dunbar-Southlands',
           'Victoria-Fraserview', 'Riley Park', 'Marpole', 'Killarney', 'Kerrisdale', 'Kitsilano',
           'Grandview-Woodland', 'Arbutus-Ridge', 'Mount Pleasant', 'Shaughnessy', 'Oakridge', 'West Point Grey',
           'Fairview', 'Strathcona', 'South Cambie', 'Downtown', 'West End']
for i in regions:
    df = pd.read_csv('cleaned.csv')
    df = df.drop(columns=['CIVIC_NUMBER', 'PCOORD', 'P_PARCEL_ID', 'SITE_ID', 'Geom', 'STD_STREET', 'full_add', 'road'])
    df = df.dropna(subset=['area', 'bathroom', 'bedroom', 'built_year', 'garage', 'total_value'])
    # select only residential buildings
    df = df[df['bedroom']!=0]
    # select district
    df = df[df['Geo Local Area']==i]


    X = df[['area', 'bathroom', 'bedroom', 'built_year', 'garage']]
    y = df['total_value']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

    model = LinearRegression(normalize=True)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print('r2:', r2_score(y_test, y_pred))
    print('MSE:', mean_squared_error(y_test, y_pred))
    dump(model, f'{i}.joblib')
```

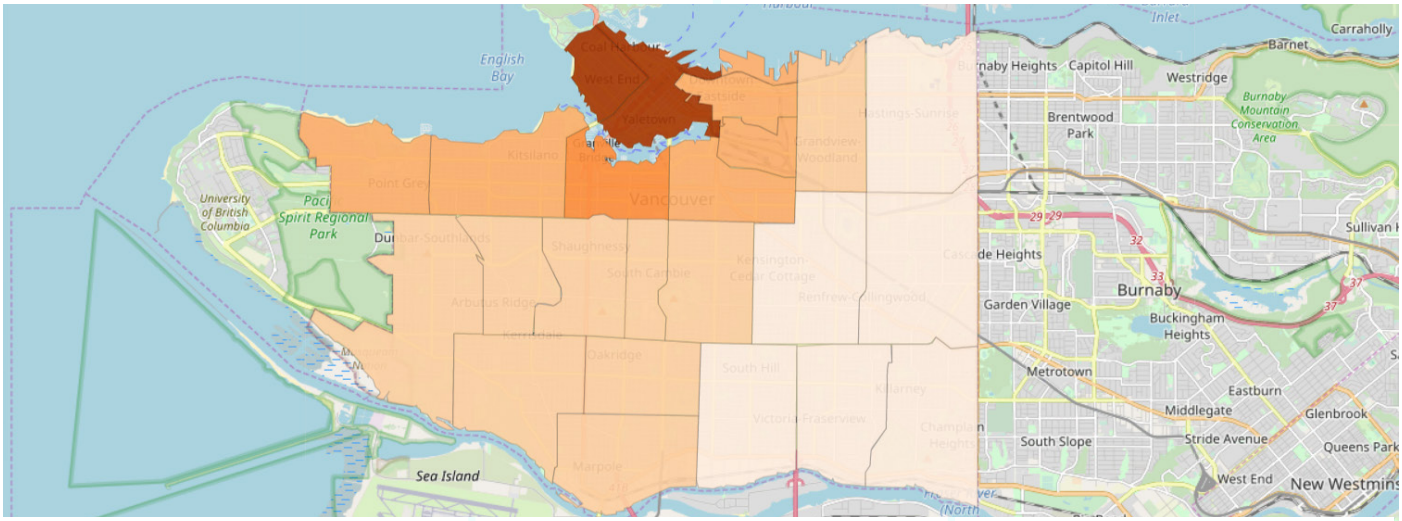
'Kensington-Cedar Cottage'	r2: 0.7275584968812627
'Renfrew-Collingwood'	r2: 0.6931773391644624
'Sunset',	r2: 0.6640833399613051
'Hastings-Sunrise'	r2: 0.7039210785582909
'Dunbar-Southlands'	r2: 0.5248994260834603
'Victoria-Fraserview'	r2: 0.7951816813279151
'Riley Park'	r2: 0.5231485037848531
'Marpole'	r2: 0.5034834962191344
'Killarney'	r2: 0.7467653870876554
'Kerrisdale'	r2: 0.6343728338318276
'Kitsilano'	r2: 0.53960635401618357
'Grandview-Woodland'	r2: 0.6680653403151944
'Arbutus-Ridge'	r2: 0.677139264661547
'Mount Pleasant'	r2: 0.47024284999609056
'Shaughnessy'	r2: 0.6471343339602749
'Oakridge'	r2: 0.46939780063189407
'West Point Grey'	r2: 0.9272779404320457
'Fairview'	r2: 0.5039335747370566
'Strathcona'	r2: 0.4826511648111762
'South Cambie'	r2: -0.449888240728034
'Downtown'	r2: -0.5364194132174658
'West End'	r2: -0.04573413199054177

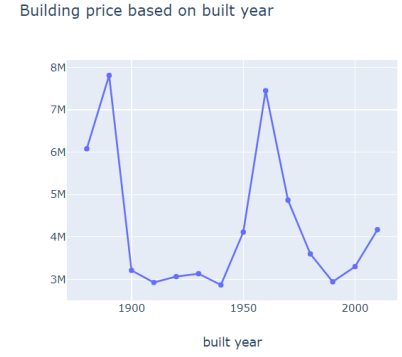
R2 score for the last three regions is negative since the dataset after filtering of these three regions is super small. For South Cambie and West End, we only have below 50 rows. For Downtown, we only have 4 rows. So the model will not be accurate for the last three regions since the model is built for predicting house price, but there aren't many houses there.

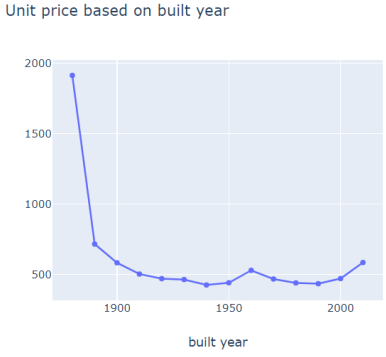
VISUALIZATION

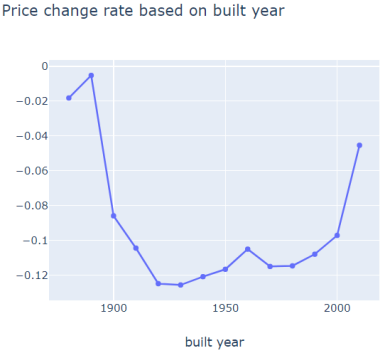
Build a dashboard using plotly-dash to present the analysis. Plot price, unit price and price change rate group by regions, the number of rooms, and binned built years.











Summary and Future Improvement

Dashboard url: http://bit.ly/vancouver-house-price

From graphs in the dashboard, we can have several assumptions:

1. For average prices in a region, only prices of buildings in Downtown increases. Houses in Shaughnessy has the largest depreciation rate compred with other regions.
2. Unit price of houses in west regions is higher than unit price in east regions. Downtown has the highest unit price, and regions next to Downtown have relative higher unit prices compared with other regions. Houses in West Point Grey (top left region) has relative higher unit price because it is close to UBC
3. Unit price of houses with 6 to 8 bathrooms is higher
4. Houses with 6 bathrooms have the lowest depreciation rate compared with houses with other number of bathrooms.
5. Generally, price, unit price, and price change rate of buildings built before 1990 or after 2000 are higher except for building built between 1960 and 1970

For future improvement, I can use selenium to search for price for one unit in condo instead of price for whole building to get more insight on Vancouver condo market.