

# **CSC 59867 - Senior Design II**

## **Fall 2019**

### **Project Delivery**

John Chen  
Abdur Rafey  
Praveena Shrestha  
Stanley Wong

# TABLE OF CONTENTS

<b>INSTALLATION GUIDE</b>	<b>2</b>
Android Studio	2
macOS	2
Windows	4
Android Virtual Device (AVD)	6
Jupyter Notebook	9
macOS	9
Python3:	9
Jupyter Notebook:	10
Windows	11
Python3:	11
Anaconda:	12
Jupyter Notebook:	12
Import Project Files	13
Android Studio	13
Jupyter Notebook	17
<b>DOCUMENTATION</b>	<b>20</b>
Android Studio	20
Activities	20
Adapters	24
Fragments	25
Receivers	27
Services	28
Utils	30
Jupyter Notebook	34

# INSTALLATION GUIDE

## Android Studio

The following are the system requirements, instructional steps to install **Android Studio** on either macOS or Windows, and how to properly import the project files onto **Android Studio**.

### macOS

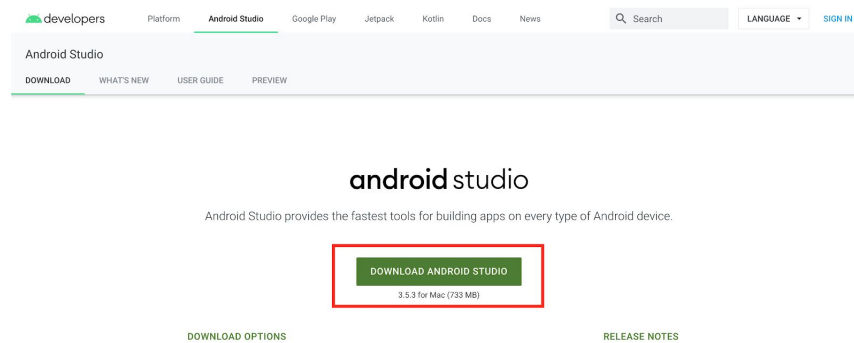
---

To install **Android Studio** on your Mac, you need to have the following system requirements:

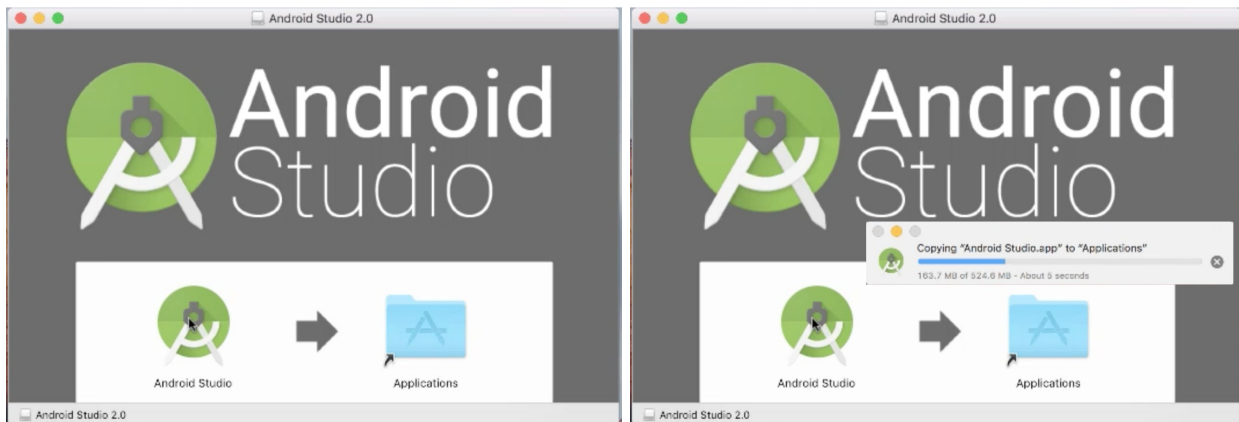
- Mac® OS X® 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave)
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Once your computer meets these system requirements, follow these steps to install and setup **Android Studio** on your Mac:

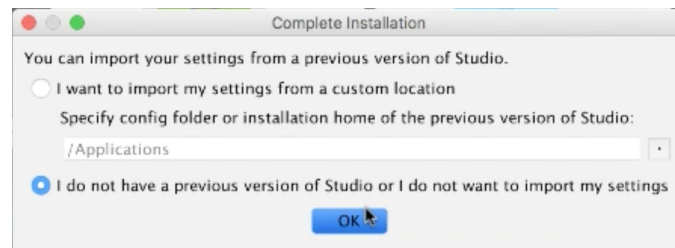
1. Go to <https://developer.android.com/studio/index.html>
2. Click the **DOWNLOAD ANDROID STUDIO** button



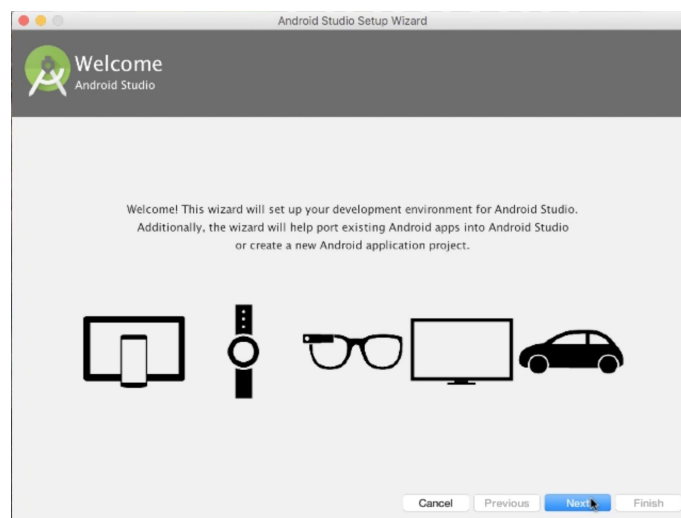
3. Launch the **Android Studio .dmg** file by double clicking the file.
4. Drag and drop **Android Studio** into the Applications folder, then launch **Android Studio**.



5. Select whether you want to import previous **Android Studio** settings, then click **OK**.



6. The **Android Studio Setup Wizard** guides you through the rest of the setup, which includes downloading Android SDK components that are required for development.



## Windows

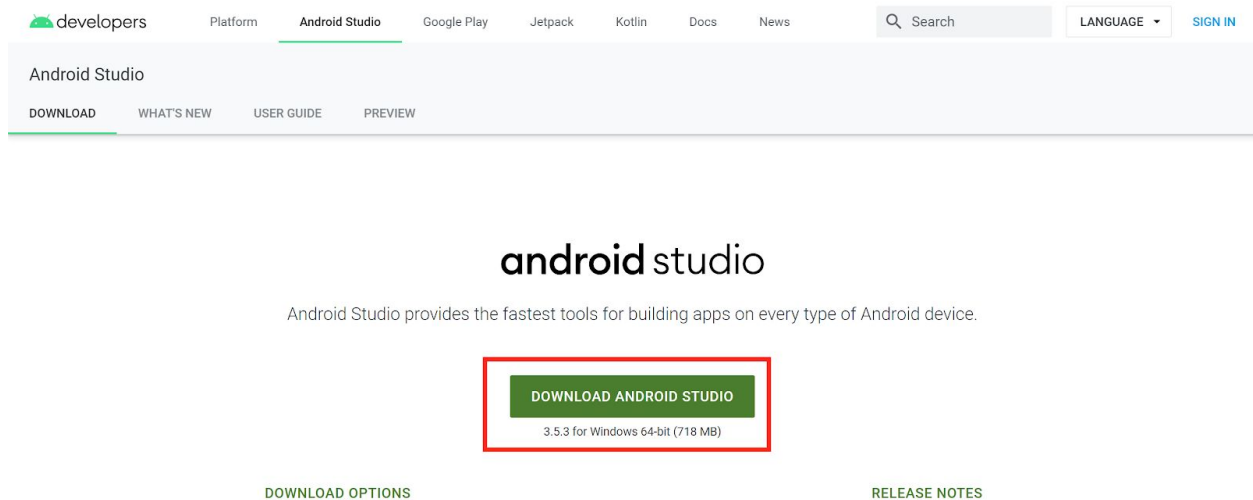
---

To install **Android Studio** on your Windows, you need to have the following system requirements:

- Microsoft® Windows® 7/8/10 (32- or 64-bit)  
*The Android Emulator only supports 64-bit Windows.*
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,  
4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

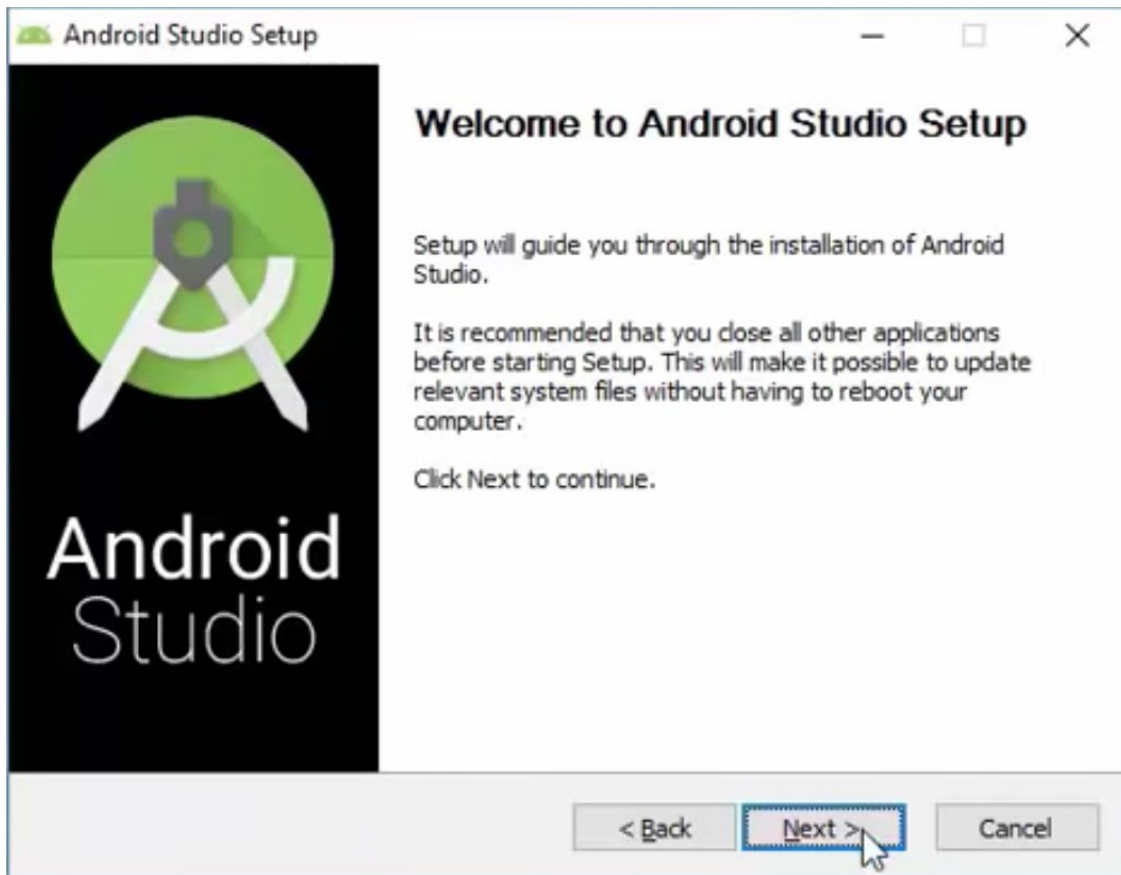
Once your computer meets these system requirements, follow these steps to install and setup **Android Studio** on your Windows:

1. Go to <https://developer.android.com/studio/index.html>
2. Click the **DOWNLOAD ANDROID STUDIO** button



3. Double click the **Android Studio** .exe file to launch the setup wizard.

4. Follow the setup wizard in **Android Studio** and install any SDK packages that it recommends.




---

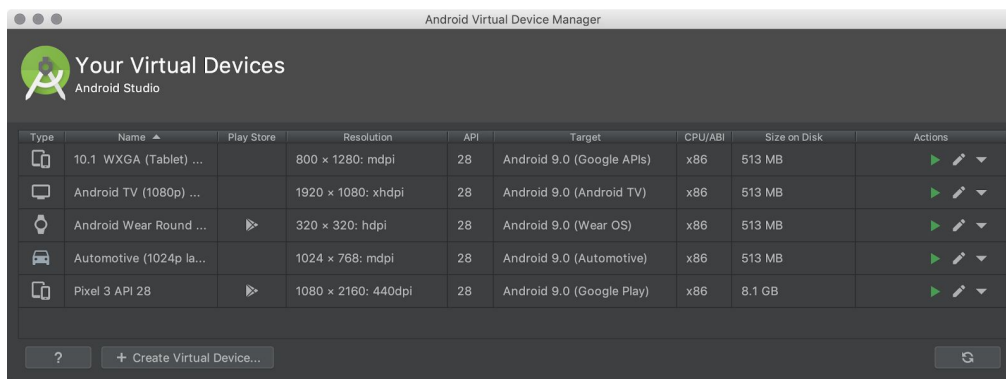
For extra information on the setup process, please refer to the setup manual from the Android Studio developers ([Learn More](#)).

## Android Virtual Device (AVD)

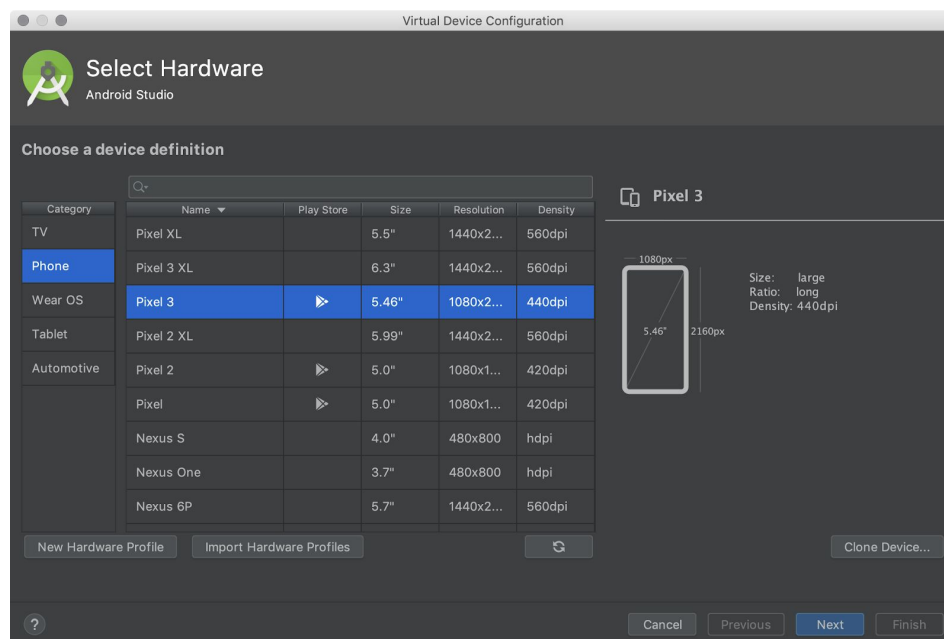
An **Android Virtual Device (AVD)** is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the [Android Emulator](#). The **AVD Manager** is an interface you can launch from **Android Studio** that helps you create and manage AVDs.

To create a new **AVD**, do the following:

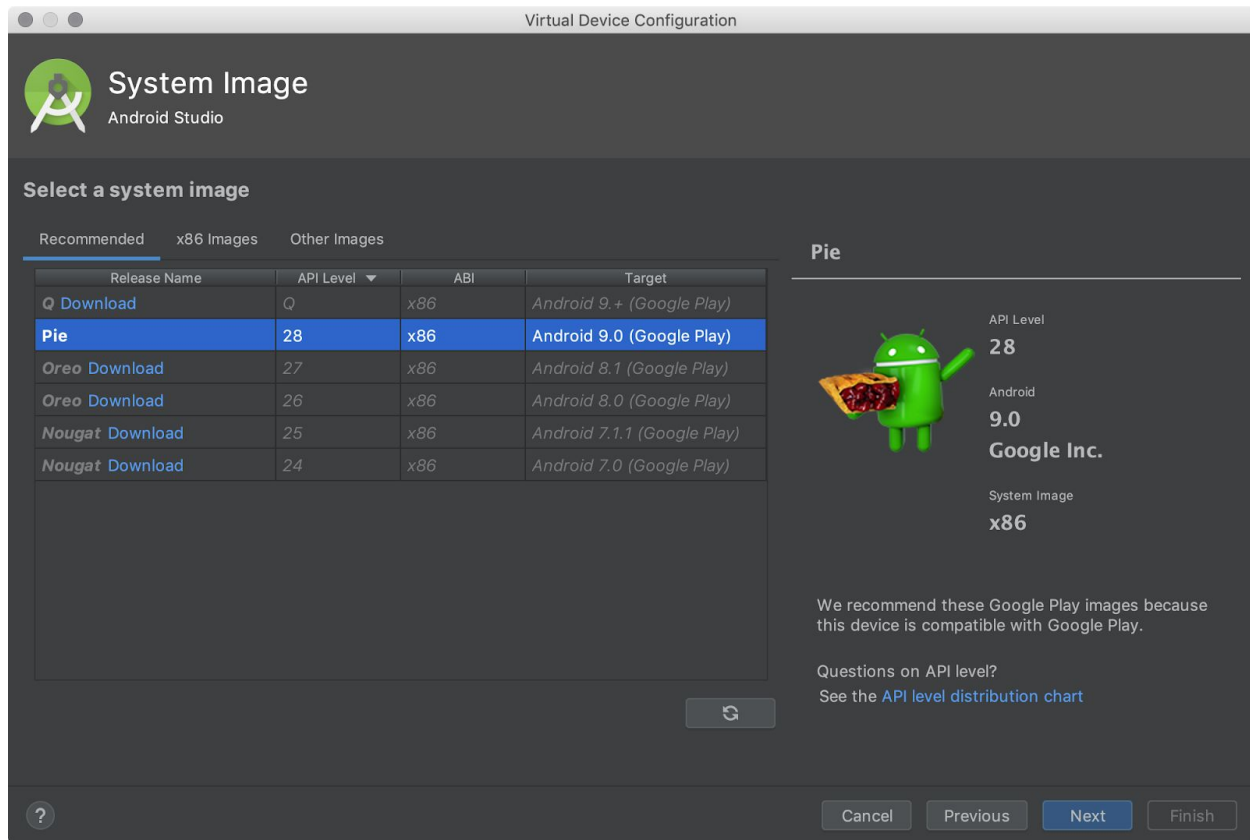
1. Select **Tools > AVD Manager** or click **AVD Manager**  in the toolbar.



2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog. The **Select Hardware** page appears.



3. Select a hardware profile, and then click **Next**. If you don't see the hardware profile you want, you can [create](#) or [import](#) a hardware profile. The **System Image** page appears.



4. Select the system image for a particular API level, and then click **Next**.

The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

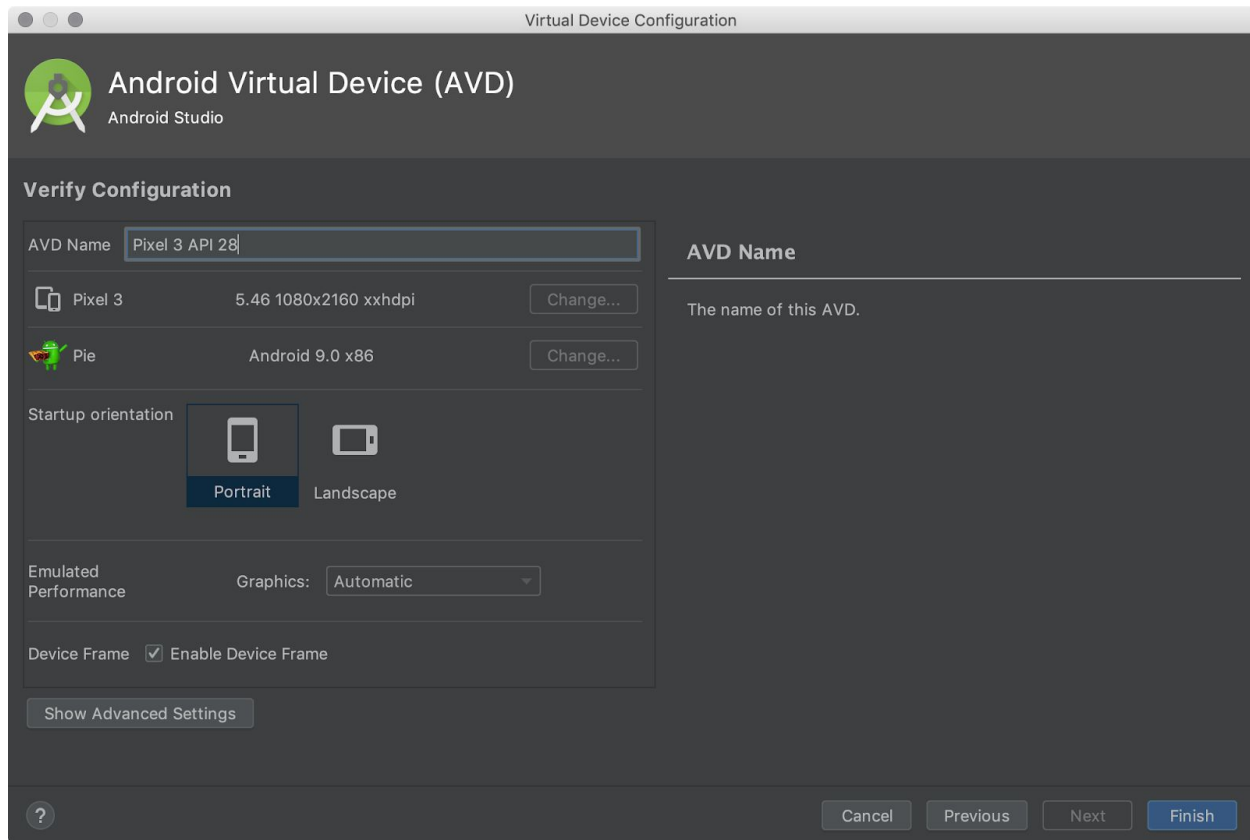
The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the `minSdkVersion` attribute of the app manifest file. For more information about the relationship between system API level and `minSdkVersion`, see [Versioning Your Apps](#).

If your app declares a `<uses-library>` element in the manifest file, the app requires a system image in which that external library is present. If you want to run your



app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The **Verify Configuration** page appears.



5. Change [AVD properties](#) as needed, and then click **Finish**.

Click **Show Advanced Settings** to show more settings, such as the skin.

The new AVD appears in the **Your Virtual Devices** page or the **Select Deployment Target** dialog.

For extra information on the AVD setup process, please refer to the AVD setup manual from the Android Studio developers ([Learn More](#)).

## Jupyter Notebook

To install Jupyter Notebook, you need to install **Python3** and **Anaconda** beforehand. The following are the system requirements, instructional steps to install **Python3** and **Anaconda** on either macOS or Windows.

**NOTE:** We need to install **Anaconda** only on Windows devices. If it is a macOS device, we do not require **Anaconda**.

### macOS

---

#### Python3:

To install Python on your macOS device, you need to have the following system requirements:

- **Jupyter** installation requires Python 3.3 or greater, or Python 2.7.

Once your system meets these requirements, follow these steps to install **Python3**:

1. Install **homebrew**
2. Go to **Terminal**
  - a. Copy and run:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

3. Then run the following into the **Terminal**:

```
brew install python3
```

## Jupyter Notebook:

Once your computer meets these system requirements, follow these steps to install and setup Jupyter Notebook on your macOS device:

1. Open **Terminal** and run:

```
sudo easy_install pip
```

2. Since **Python3** is pre-installed, run the following command in the **Terminal**:

```
python3 -m pip install --upgrade pip  
python3 -m pip install jupyter
```

3. To open Jupyter Notebook, run the following command in the **Terminal**:

```
jupyter notebook
```

## Windows

---

### Python3:

To install Python3 on your Windows device, you need to have the following system requirements:

- **Jupyter** installation requires Python 3.3 or greater, or Python 2.7.

Once your system meets these requirements, follow these steps to install **Python3**:

1. Go to <https://www.python.org/downloads/release/python-365/>
2. Choose any python windows installer and finish installation:

<a href="#">Windows x86-64 embeddable zip file</a>	Windows	for AMD64/EM64T/x64	04cc4f6f6a14ba74f6ae1a8b685ec471	7190516	<a href="#">SIG</a>
<a href="#">Windows x86-64 executable installer</a>	Windows	for AMD64/EM64T/x64	9e96c934f5d16399f860812b4ac7002b	31776112	<a href="#">SIG</a>
<a href="#">Windows x86-64 web-based installer</a>	Windows	for AMD64/EM64T/x64	640736a3894022d30f7babff77391d6b	1320112	<a href="#">SIG</a>
<a href="#">Windows x86 embeddable zip file</a>	Windows		b0b099a4fa479fb37880c15f2b2f4f34	6429369	<a href="#">SIG</a>
<a href="#">Windows x86 executable installer</a>	Windows		2bb6ad2ecca6088171ef923bca483f02	30735232	<a href="#">SIG</a>
<a href="#">Windows x86 web-based installer</a>	Windows		596667cb91a9fb20e6f4f153f3a213a5	1294096	<a href="#">SIG</a>

3. Go to **Command Prompt** and run following code to check:

```
python --version
```

**Anaconda:**

To install Anaconda on your Windows, you need to have the following system requirements:

- Windows- 64-bit x86, 32-bit x86.
- Minimum 5 GB disk space to download and install.

Once your computer meets these system requirements, follow these steps to install and setup **Anaconda** on your Windows device:

1. Go to <https://www.anaconda.com/distribution/#download-section>
2. Click **DOWNLOAD** on the **Python 3.7 VERSION**

**Anaconda 2019.10 for Windows Installer**

Python 3.7 version	Python 2.7 version
<a href="#">Download</a>	<a href="#">Download</a>
64-Bit Graphical Installer (462 MB)	64-Bit Graphical Installer (413 MB)
32-Bit Graphical Installer (410 MB)	32-Bit Graphical Installer (356 MB)

3. Install the version of the downloaded Anaconda:

**Jupyter Notebook:**

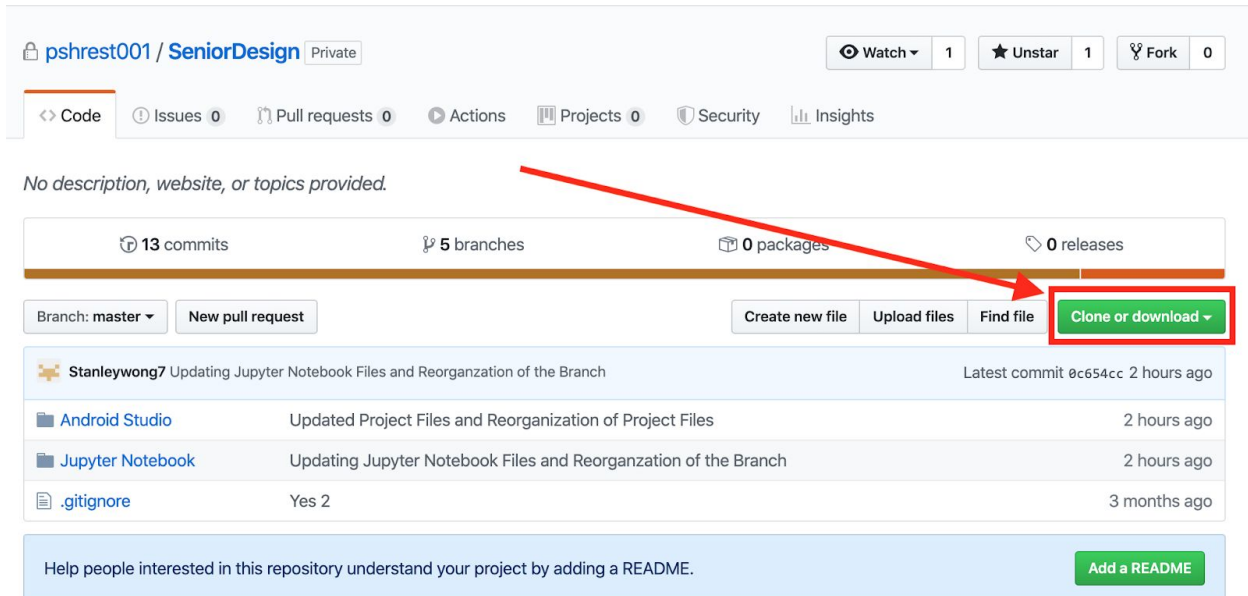
1. To open Jupyter Notebook, run following in **Command Prompt**:

```
jupyter notebook
```

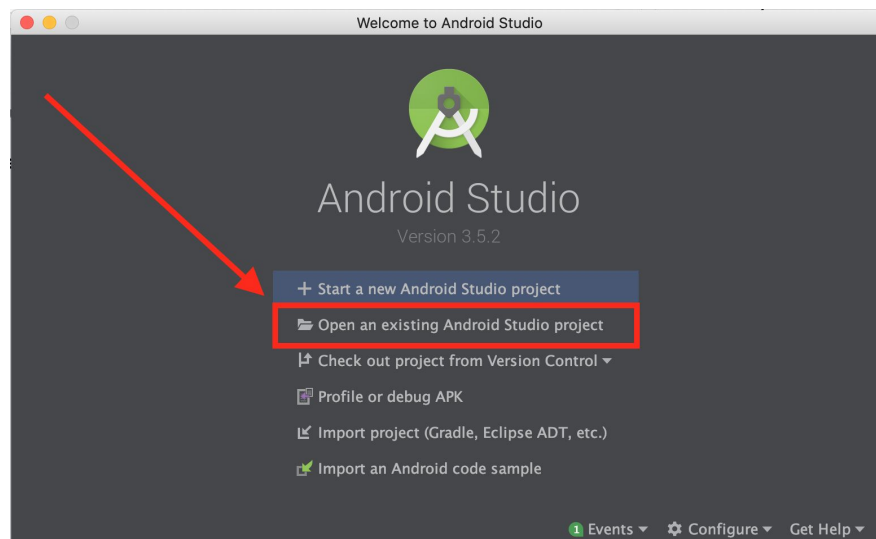
# Import Project Files

## Android Studio

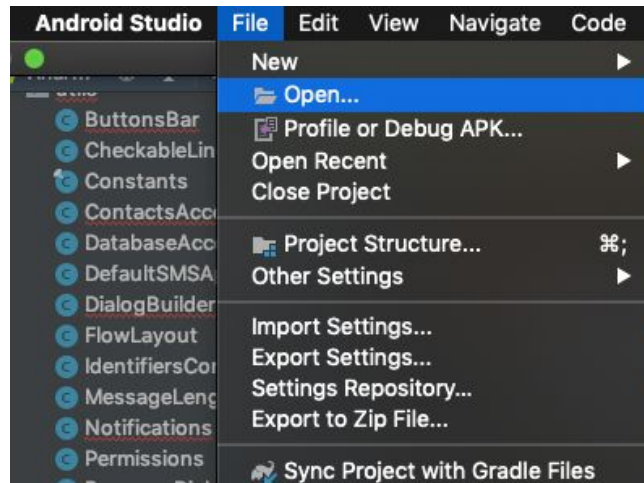
1. Go [here](#) and click on the **Clone or Download** button.



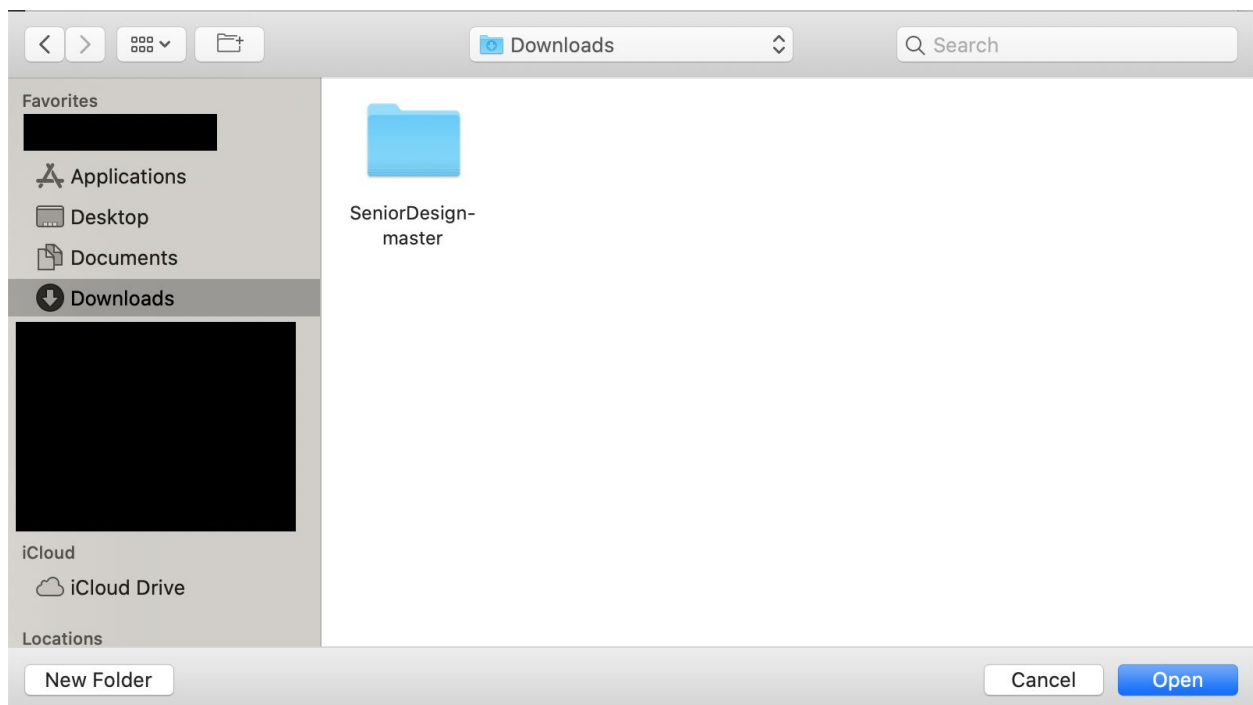
2. Go to your **Downloads** folder on your computer.
3. Double click on the .zip **SeniorDesign-master** file to unzip it.
4. Open **Android Studio** and click on **Open and existing Android Studio project**



5. If you have an existing Android Studio project open, go to **File >> Open...** If not, ignore this step.

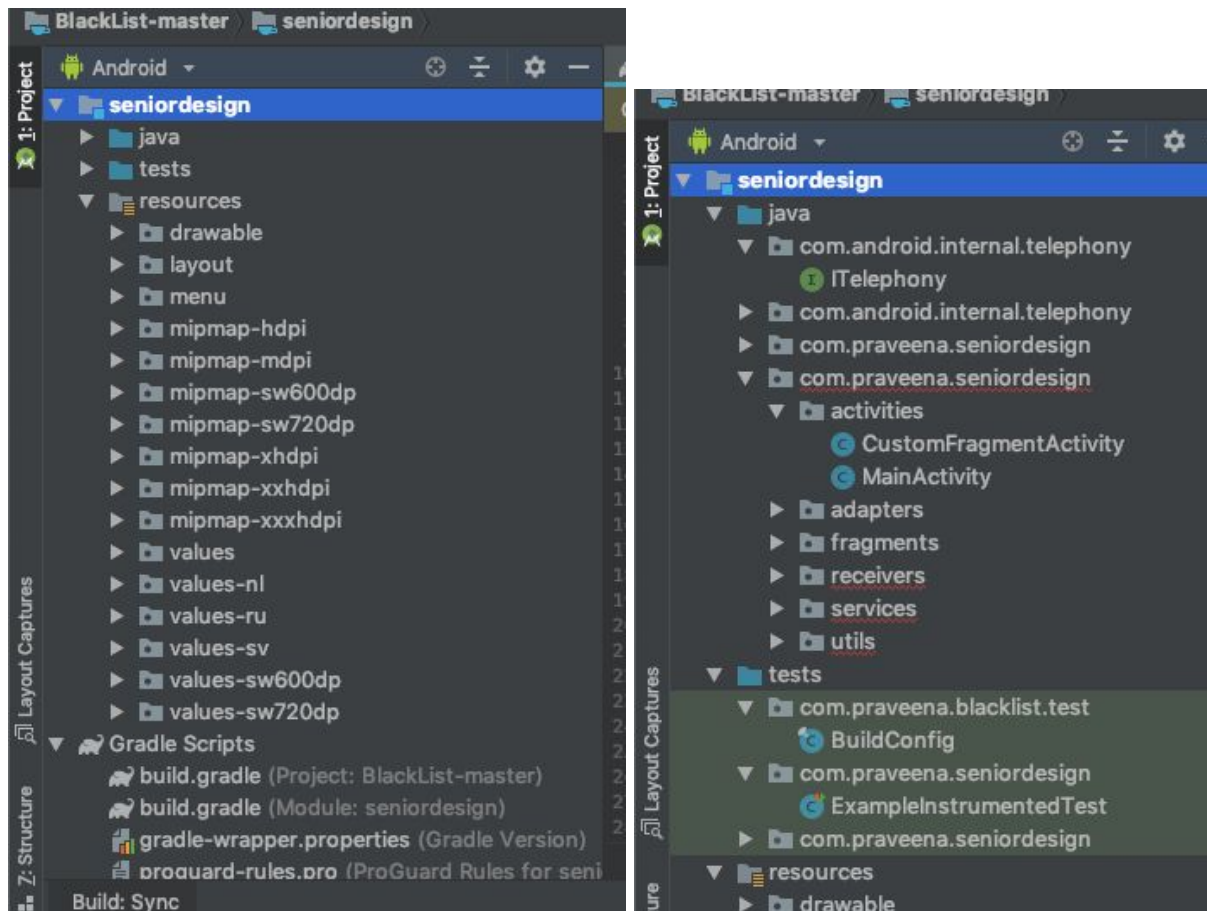


6. Inside the window, navigate through your computer's directory to go to your **Downloads** folder to find **SeniorDesign-master** folder.



7. Select the **SeniorDesign-master** folder and click **Open**.

8. On the left side, you should be able to see these folders and click the sub folders such as java, test in order to see all the branches and the subfolders as shown in the right side. These all subfolders are described in the documentation.



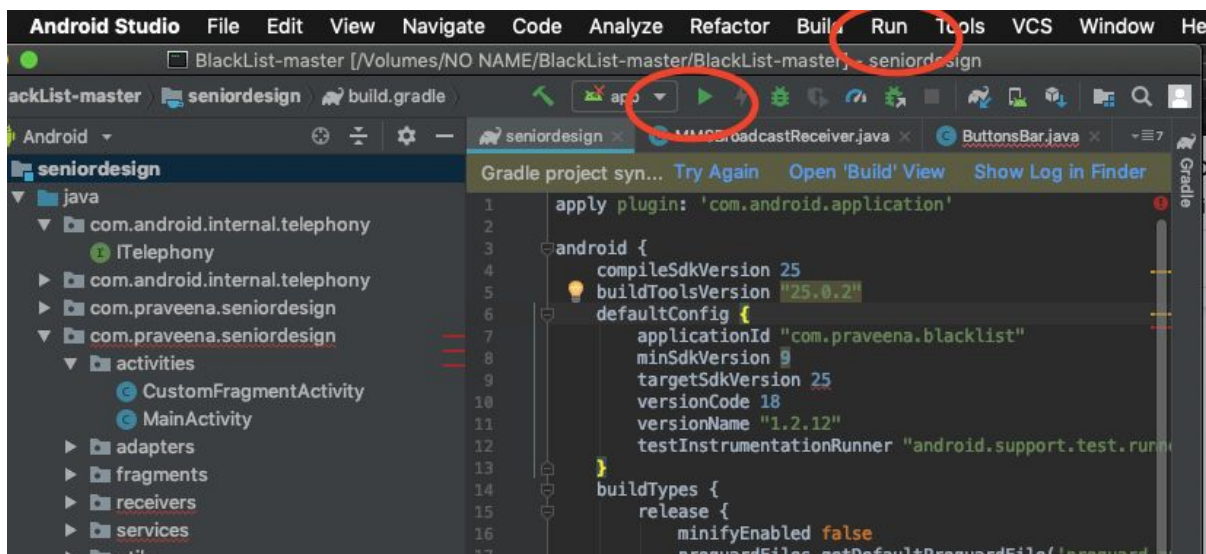
9. You have to go to **build.gradle** first and sync the project, which shows the pop up the top of the screen like below:



10. On the project folder directory, go to  
**java >> com.praveena.seniordesign >> activities >> mainactivity.java**



11. Run the app by going to the **Tools** tab, which is on the top or side bar as shown below.



12. It has to show the execution being all the way done in the terminal and show build in progress.
13. You will have to **download** AVD Manager which stands for android virtual devices in order to see the app running. To download the virtual device, please refer to the **Android Virtual Device (AVD)** section of the manual.
14. You can choose any version of android and download any pixel devices or tablet as your choice.

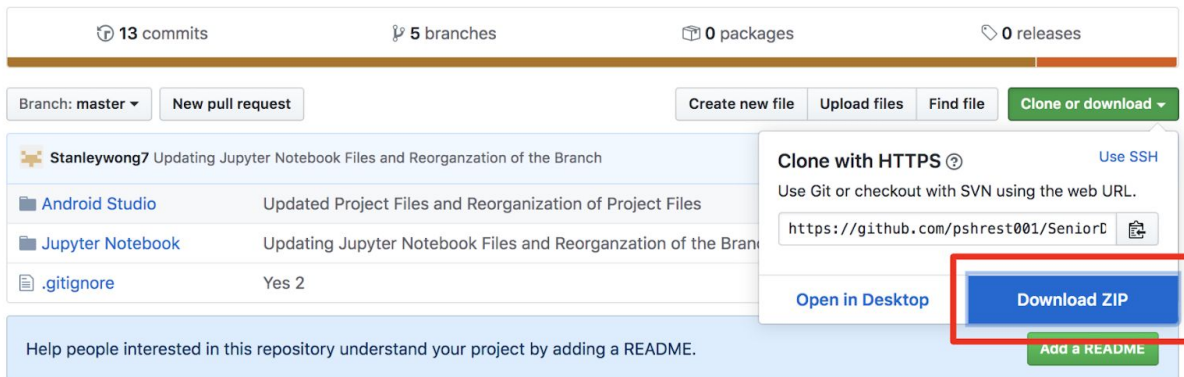


15. You can press the play button and the android emulator will be able to show our app called "BlackList"

## Jupyter Notebook

1. Go [here](#) to download the github project and unpackage the zip folder:

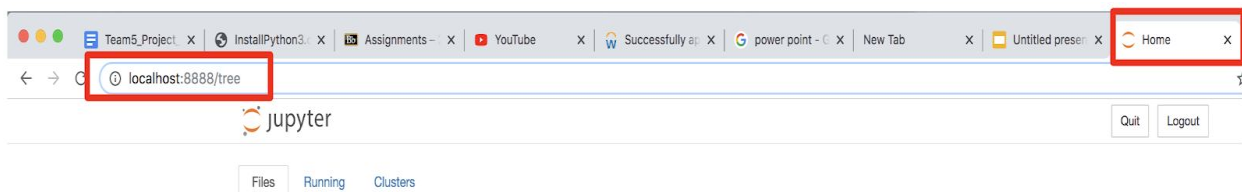
*No description, website, or topics provided.*



2. To open Jupyter Notebook, run following in **Command Prompt** or **Terminal**:

```
jupyter notebook
```

3. The Jupyter Notebook web application will now be opened in your browser:
  - a. You will see a tab labelled **Home**



4. Navigate to the **Downloads** folder link and click on it:



The screenshot shows the JupyterLab interface with the file browser on the left. The breadcrumb path is "/". The file list includes:

Name	Last Modified	File size
anaconda3	6 months ago	
Applications	5 months ago	
Desktop	2 days ago	
Documents	3 months ago	
<b>Downloads</b>	<b>a day ago</b>	
Flix_demo_03	a year ago	
google-cloud-sdk	4 days ago	
homebrew	10 months ago	
https:	9 months ago	

5. Navigate to Machine Learning files:


a. Find the **SeniorDesign-master** folder link and click on it:



The screenshot shows the file browser with the breadcrumb path "/ Downloads /". The file list includes:

Name	Last Modified	File size
SE_fproj32 copy	7 months ago	
SE_fproj4	7 months ago	
SE_fprojsearchbar	7 months ago	
<b>SeniorDesign-master</b>	<b>15 minutes ago</b>	
Shopping_cart-master	7 months ago	
TipCalculatorStarter-master 2	a year ago	
Tutorial_JFLAP_Files	12 years ago	

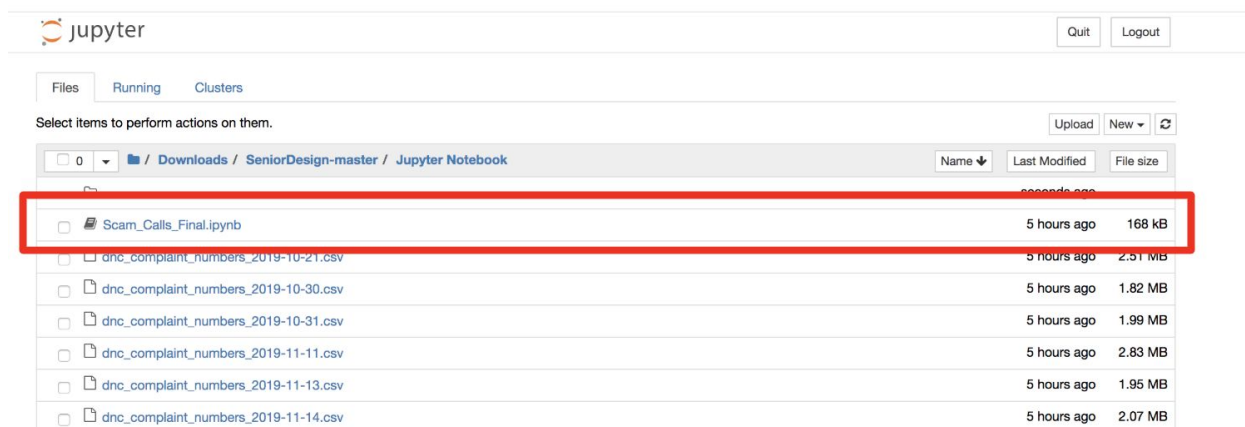
b. Click on the **Jupyter Notebook** folder link



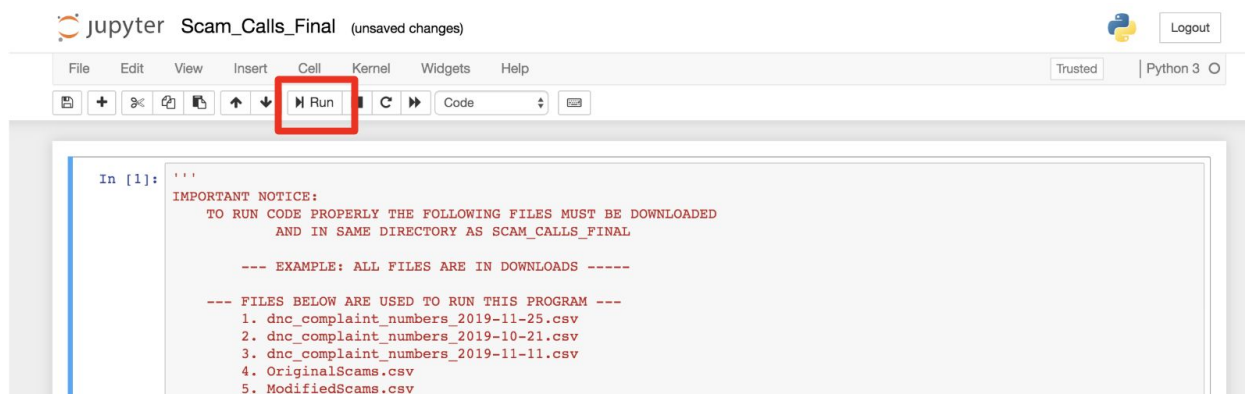
The screenshot shows the file browser with the breadcrumb path "/ Downloads / SeniorDesign-master". The file list includes:

Name	Last Modified	File size
..	seconds ago	
Android Studio	5 hours ago	
<b>Jupyter Notebook</b>	<b>10 minutes ago</b>	

6. Proceed to click on the **Scam\_Calls\_Final.ipynb** file and it will open in a new page



7. Click on the **run** button to compile the code within each cell **OR** scroll down to see already generated code, if there is:
- IMPORTANT:** You must **run** from the very first cell to the last, if you try to run somewhere in the middle, you will get error messages of things being undefined.
  - To access a cell, simply click within the gray box area



# DOCUMENTATION

## Android Studio

The following is an explanation of each file's functionality and usage in the **Android Studio** project found categorized by the folder name. The activities subsection will detail explanations from a line-to-line code perspective, as it is the main file that integrates the entire project files. The subsections following it will give brief summaries of each file's functionalities.

### Activities

---

The directory to this folder is `app/src/main/java/com.example.myapplication/activities`. In this folder, there are two class files: `CustomFragmentActivity` and `MainActivity`. The activities class is the entry point for interacting with the user. It represents a single screen with a user interface.

```
public class CustomFragmentActivity extends AppCompatActivity
```

`CustomFragmentActivity` creates a fragment, extends the `Fragment` class to `AppCompatActivity`, then overrides key lifecycle methods to insert the app's logic.

```
@Override  
protected void onCreate(Bundle savedInstanceState)
```

In this code snippet (found on line 32), the `onCreate()` function is used to initialize the activity. When the activity starts, it will load the menu and toolbars on the app and will account the view for the screen rotation.

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    if (item.getItemId() == android.R.id.home) {  
        finish();  
        return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

The `onOptionsItemSelected` function (found on line 82) switches to the selected item in the app's menu bar once the app detects the user's interaction.

```
import com.praveena.seniordesign.R;
import com.praveena.seniordesign.fragments.ContactsFragment;
import com.praveena.seniordesign.fragments.FragmentArguments;
import com.praveena.seniordesign.fragments.InformationFragment;
import com.praveena.seniordesign.fragments.JournalFragment;
import com.praveena.seniordesign.fragments.SMSConversationFragment;
import com.praveena.seniordesign.fragments.SMSConversationsListFragment;
import com.praveena.seniordesign.fragments.SMSSendFragment;
import com.praveena.seniordesign.fragments.SettingsFragment;
import com.praveena.seniordesign.utils.ContactsAccessHelper;
import com.praveena.seniordesign.utils.DatabaseAccessHelper.Contact;
import com.praveena.seniordesign.utils.Permissions;
import com.praveena.seniordesign.utils.Settings;
```

*MainActivity* is the main file that integrates all fragment files to work together alongside the user interface. The following code snippet, shown above, is part of the file's header (found in line 23) to import all the fragment class files, found within the *Fragment* folder, and the utils class files, found within the *Utils* folder.

```
protected void onCreate(Bundle savedInstanceState) {
```

This function can be found in line 52. Similar to the `onCreate()` function in the *CustomFragmentActivity* file, it also initializes the activity once the user opens the application. This function's logic is the same as the logic found in *CustomFragmentActivity*; however, it also includes the switch cases for each action the user performs, shown below.

```

switch (action) {
    case ACTION_SMS_SEND_TO:
        // show SMS sending activity
        showSendSMSActivity();
        // switch to SMS chat fragment
        itemId = R.id.nav_sms;
        break;
    case ACTION_SMS_CONVERSATIONS:
        // switch to SMS chat fragment
        itemId = R.id.nav_sms;
        break;
    case ACTION_SETTINGS:
        // switch to settings fragment
        itemId = R.id.nav_settings;
        break;
    case ACTION_JOURNAL:
        // switch to journal fragment
        itemId = R.id.nav_journal;
        break;
    default:
        if (Settings.getBooleanValue(this, Settings.GO_TO_JOURNAL_AT_START)) {
            // switch to journal fragment
            itemId = R.id.nav_journal;
        } else {
            // switch to SMS chat fragment
            itemId = R.id.nav_sms;
        }
        break;
}

```

The switch cases (found in line 95) tells the application which fragment to run the application's last state. So for example, if the user closed the app when it was in the SMS Chat menu, if the app is still running in the background, it can continue where the user left off.

```
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
```

Found in line 152, `onNavigationItemSelected()` function tells the program what the user has tapped on and where to direct that action to which fragment. Normally we don't need to select navigation items manually. But in API 10 there is bug of menu item selection/deselection. To resolve this problem we deselect the old selected item and select the new one manually. And it's why we return false in the current method. This way of deselection of the item was found as the most appropriate. Because of some side effects of all others tried.

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String permissions[],
                                       @NonNull int[] grantResults) {

    // process permissions results
    Permissions.onRequestPermissionsResult(requestCode, permissions, grantResults);
    // check granted permissions and notify about not granted
    Permissions.notifyIfNotGranted(this);
}
```

Found in line 185, `onRequestPermissionsResult()` function requests the user for certain permissions if the mobile application was freshly installed and ran the program for the first time on the Android device.

```
private void showSendSMSActivity() {
```

Found in line 296, `showSendSMSActivity()` function will give the user the ability to view and use their SMS text messages on the mobile application, once granted the app SMS Permissions. The function also allows the user to view blocked messages on the app.

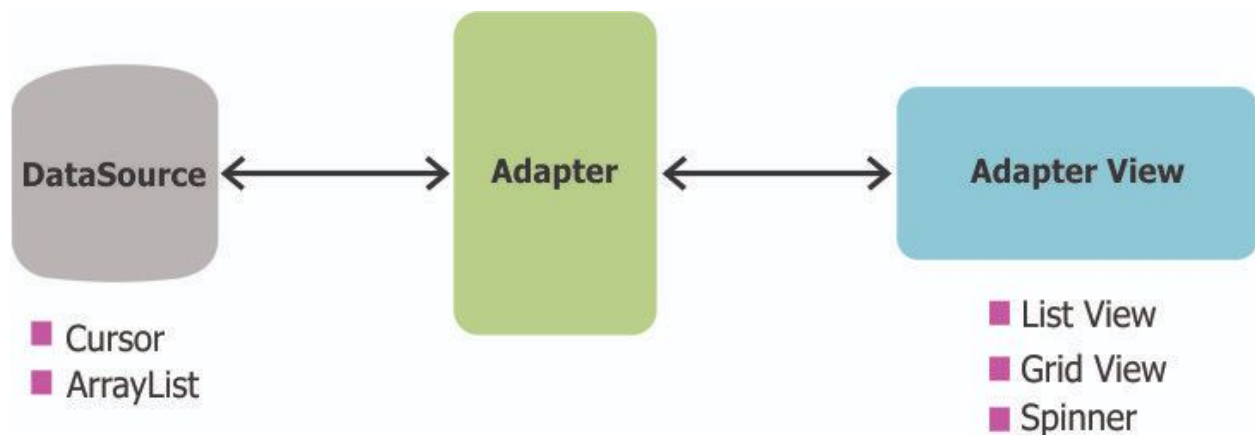


## Adapters

---

The directory to this folder is `app/src/main/java/com.example.myapplication/adapters`. In this folder, there are six files: *ContactsCursorAdapter*, *InformationArrayAdapter*, *JournalCursorAdapter*, *SettingsArrayAdapter*, *SMSConversationCursorAdapter*, and *SMSConversationListCursorAdapter*.

Adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to an Adapter view then view can take the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc. To visualize how Adapters work, please refer to the following diagram below.



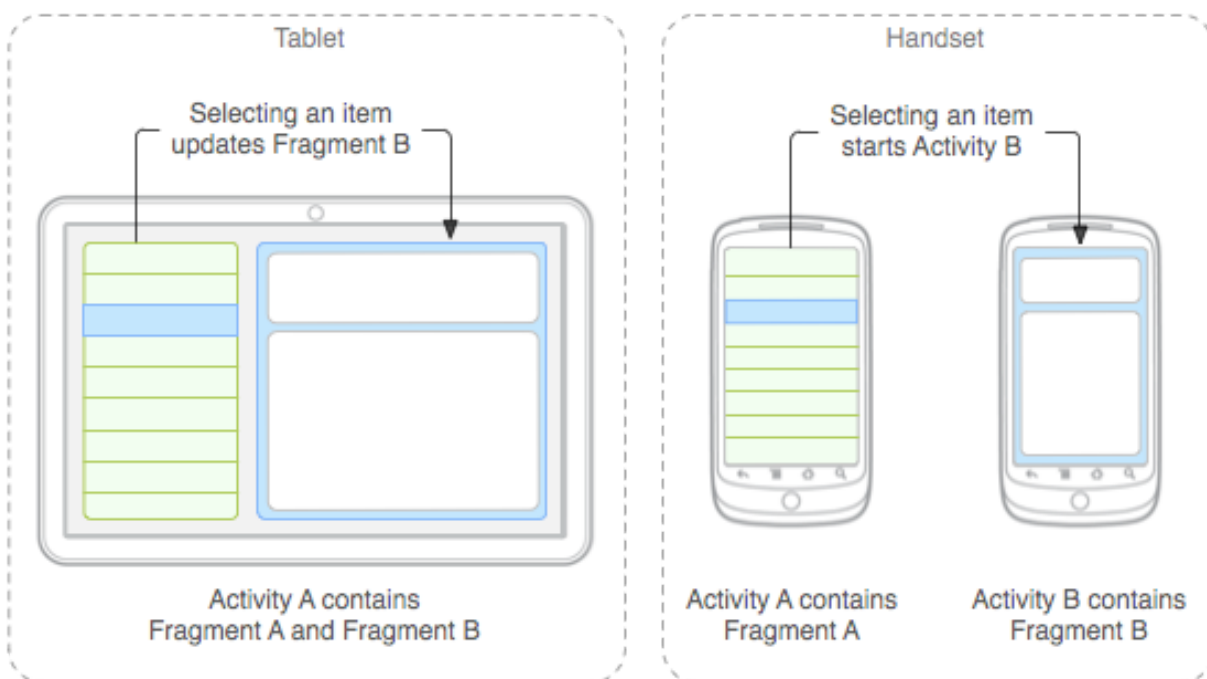
*CursorAdapter* is an adapter that exposes the user's data from a `Cursor` to a `ListView` widget. *ContactsCursorAdapter* helps reorganize the user's contacts and put it into a list form for the user to view on the mobile application. *JournalCursorAdapter* takes the user's call log history and puts it into a list form for the user to view on the app. *SMSConversationCursorAdapter* and *SMSConversationListCursorAdapter* takes in the user's SMS text message content and history and puts it into a list form for the user to view on the app.

*ArrayAdapter* is an adapter to provide views for an `AdapterView` and returns a view for each object in a collection of data objects you provide, and can be used with list-based user interface widgets such as `ListView` or `Spinner`. *InformationArrayAdapter* takes the information blurb we provided, which details what the app is capable of and who the creators are, and puts it into a `ListView` inside the *Information* menu tab. *SettingsArrayAdapter* takes the settings information and layout we provided and puts it into a `ListView` inside the *Settings* menu tab.

## Fragments

The directory to this folder is `app/src/main/java/com.example.myapplication/fragments`. In this folder, there are eleven files: `AddContactsFragment`, `AddOrEditContactFragment`, `ContactsFragment`, `FragmentArguments`, `GetContactsFragment`, `InformationFragment`, `JournalFragment`, `SettingsFragment`, `SMSConversationFragment`, `SMSConversationsListFragment`, and `SMSSendFragment`

Fragment is a part of an activity which enable more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents a behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle. As explained in the **Activities** section, we can create Fragments by extending Fragment class or by inserting a Fragment into our Activity layout by declaring the Fragment in the activity's layout file, as a `<fragment>` element. We can manipulate each Fragment independently, such as add or remove them. The diagram below visualizes how Fragments work in Android.



*AddContactsFragment* handles the activities and functionalities of adding contacts directly from the mobile application. It is also a helper function to the *AddOrEditContactFragment* function. *AddOrEditContactFragment* handles the activities and functionalities of adding and editing contacts from the mobile application. *GetContactsFragment* handles the activities and functionalities of getting the user's contact data and importing it over the mobile application. *ContactsFragment* uses the *AddContactsFragment*, *AddOrEditContactFragment*, *GetContactsFragment*, and *ContactsCursorAdapter* to integrate them all together for the **Contacts** tab on the application's menu.

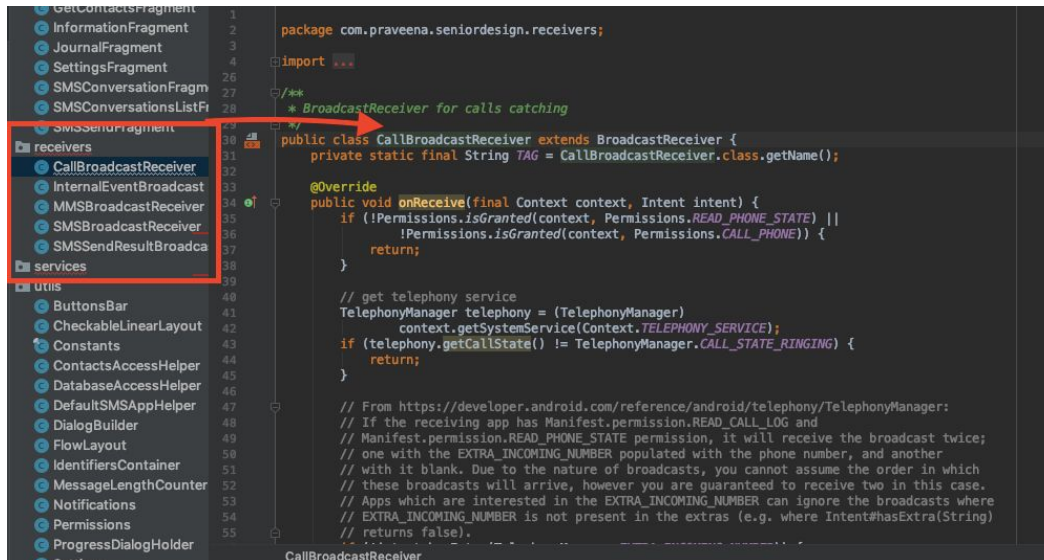
*FragmentArguments* is a common way to instantiate and pass data to fragments in **Android Studio**. So if Android decides to recreate the fragments later, it's going to call the no-argument constructor of the fragment. To learn more about this design practice, please click [here](#).

*InformationFragment* handles the activities and functionalities of displaying the application's information via the **Information** tab on the app's menu. This fragment imports *InformationArrayAdapter* to help display the data to the user on the app. *JournalFragment* handles the activities and functionalities of importing and displaying the user's call log history onto the mobile application in the **Journal** tab on the app's menu. This fragment imports *JournalCursorAdapter* to help display the call log history to the user on the app. *SettingsFragment* handles the activities and functionalities of the application's settings and user preferences in the **Settings** tab on the app's menu. This fragment imports *SettingsArrayAdapter* to display the settings to the user on the app.

*SMSConversationsListFragment* is a helper function to import the user's SMS text message history over to the mobile application. This fragment imports *SMSConversationsListFragment* to help display the text message history to the user on the app. *SMSSendFragment* is a helper function that handles the activities and functionalities of sending an SMS and MMS message on the app. *SMSConversationFragment* uses *SMSConversationsListFragment* and *SMSSendFragment* to integrate them all together for the **SMS** tab on the application menu.

## Receivers

The directory to this folder is `app/src/main/java/com.example.myapplication/receivers`. In this folder, there are five files: `CallBroadcastReceiver`, `InternalEventBroadcast`, `MMSBroadcastReceiver`, `SMSBroadcastReceiver`, and `SMSSendResultBroadcastReceiver`. `CallBroadcastReceiver`



There exists a package called `android.content.BroadcastReceiver` which helps to broadcast receiver for calls catching in the app. This file asks for a string called TAG and gets overridden by the class called `onReceive` which grants the person permission to read the phone by using `Permissions.READ_PHONE_STATE`, else it will get returned. It is all managed by android's Telephony Manager and helps get the telephony service. We have also written a description of how this service works in the comment section.

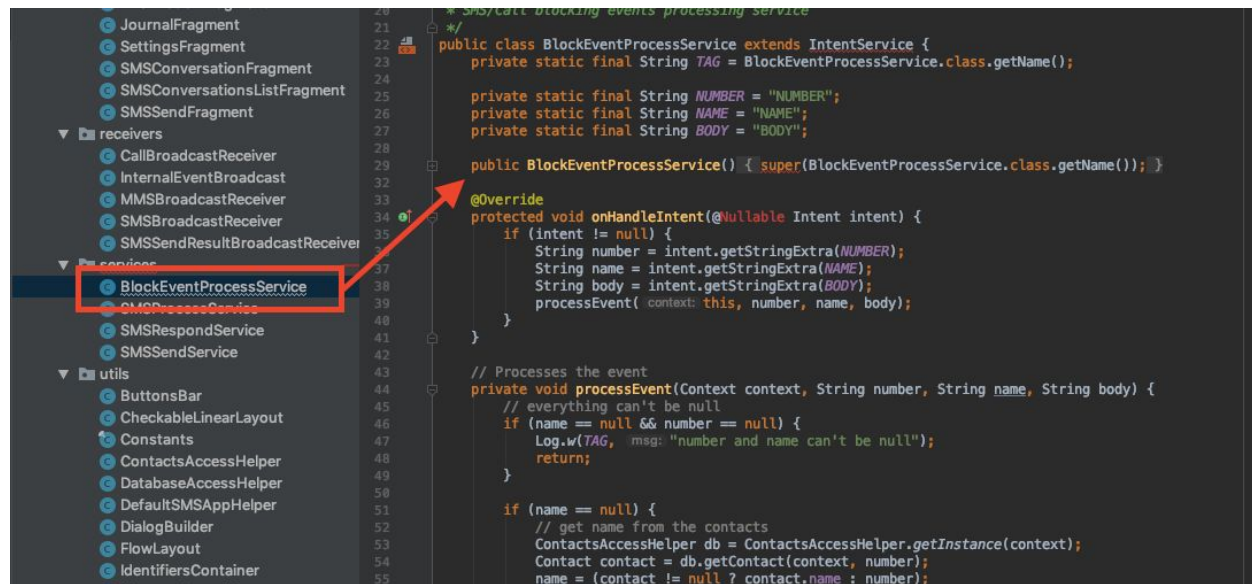
Due to the nature of broadcasts, you cannot assume the order in which these broadcasts will arrive. However, you are guaranteed to receive two in this case. Apps which are interested in the `EXTRA_INCOMING_NUMBER` can ignore the broadcasts where `EXTRA_INCOMING_NUMBER` is not present in the extras, where `Intent#hasExtra(String)` returns false. In order to break the call system from receiving, we have used a class called `breakcall` which uses the **PIE\_API\_VERSION (line 184)** with the help of SDK.

For the `InternalEventBroadcast`, we must broadcast whichever service was written and to do so we had to take the written accounts for the SMS which were read from their respective contact number and unique thread id. To invoke the callback correspondent to the received event, we used switch case statements which look for action type and reports them. For `MMSBroadcastReceiver`, `SMSBroadcastReceiver`, `SMSSendResultBroadcast`, we had to implement the java classes that were used by the previous two files.

## Services

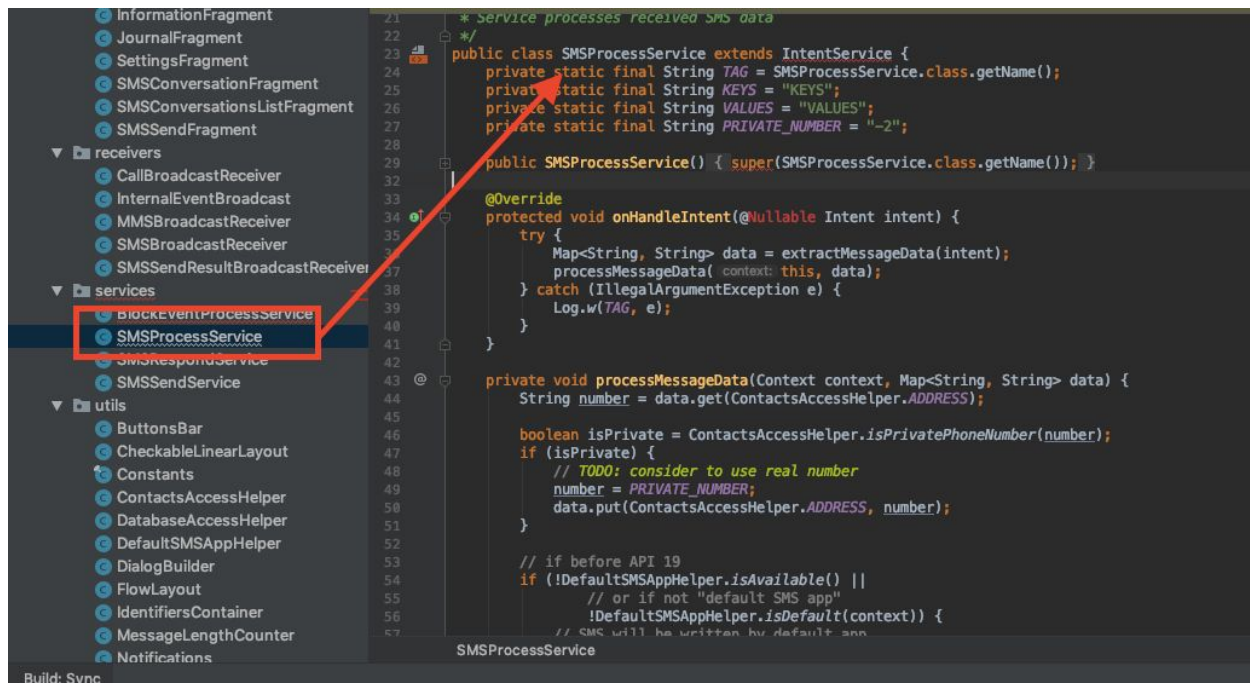
The directory to this folder is `app/src/main/java/com.example.myapplication/services`. In this folder, there are four files: *BlockEventProcessService*, *SMSProcessService*, *SMSRespondService*, and *SMSSendService*.

### Block Event Process Service File:



This function named *BlockeventProcessService* does the main basic function of the app which is SMS/Call blocking events processing service and, as shown in the figure above, it asks for the main components needed to block any number: phone number, name and body. After the file gets the name, it is passed on to the function called *onHandleIntent* which is followed by another function called “*processEvent*” which looks for the base case. The base case checks if the number and name is null and, if so, produces the error “number and name can’t be null”. Otherwise, it will use the *Notifications.onSmsBlocked* and block the contact. In order to do that, the functions *removefromcalllog* and *writetojournal* are called which helps to access the data by using *DatabaseAccessHelper.db* and deletes the content.





### SMS Process/ Receive/Send Service :

It is easier to describe all three files in one since all the files tend to use the same classes, but reverse the functionality of the service. In order to send the SMS for the android application we have used Intent service extended with the class which is part of a module package that comes with Android Studio. The first java file "SMSProcessService" requires keys, values and private number in order to process the message. We have considered using the base case of having private number with no null exception or else it will catch IllegalArgumentException error. In order to use the services of processing, receiving and sending messages, we have used **API 19 (ref. Line 53, 76)** and the DefaultSMSHelper. We have considered the numbers sending SMS showing private number should be valid. If the user wants to access the message in future then we have created the loop for each number with their unique key id and will be able to go through them and find that particular message.

All the messages get stored inside the database and can be accessed by using the *ContactsAccessHelper.db* which gets the contact by number and shows the call log in the phone app. For responding to the service, we have extended the service module called android.app.Service and used Intent and android.os.IBinder with it. This enables us to respond to all the collected messages in the application.

For the SMS Sending service, we have again used Intent service with input of message and the address. This gets overridden by onHandleIntent which first checks if the input areas have null values and then it gets message, body, and address. To send the message, *smsSendHelper.sendSMS* is called with the text that the app users have texted and it gets sent.

## Utils

---

The directory to this folder is `app/src/main/java/com.example.myapplication/Utils`. In this folder, there are seventeen files: *ButtonsBar*, *CheckableLinearLayout*, *Constants*, *ContactsAccessHelper*, *DatabaseAccessHelper*, *DefaultSMSAppHelper*, *DialogBuilder*, *FlowLayout*, *IdentifiersContainer*, *MessageLengthCounter*, *Notifications*, *Permissions*, *ProgressDialogHolder*, *Settings*, *SMSSendHelper*, *SubscriptionHelper*, and *Utils*.

### ButtonsBar:

This file customizes the snack bar which basically stands for all the buttons of the app. We have set the name for each button and changed the settings, shown below.

```
// Sets button's parameters
public boolean setButton(@IdRes int buttonId, String title, View.OnClickListener listener) {
    TextView button = (TextView) view.findViewById(buttonId);
    if (button != null) {
        button.setText(title);
        button.setOnClickListener(listener);
        button.setVisibility(View.VISIBLE);
        return true;
    }
    return false;
}

// Returns true if snack bar is shown
public boolean isShown() { return view.getVisibility() == View.VISIBLE; }

// Shows snack bar
public void show() {
    if (!isShown()) {
        view.setVisibility(View.VISIBLE);
    }
}

// Hides snack bar
public boolean dismiss() {
    if (isShown()) {
        view.setVisibility(View.GONE);
    }
}
```

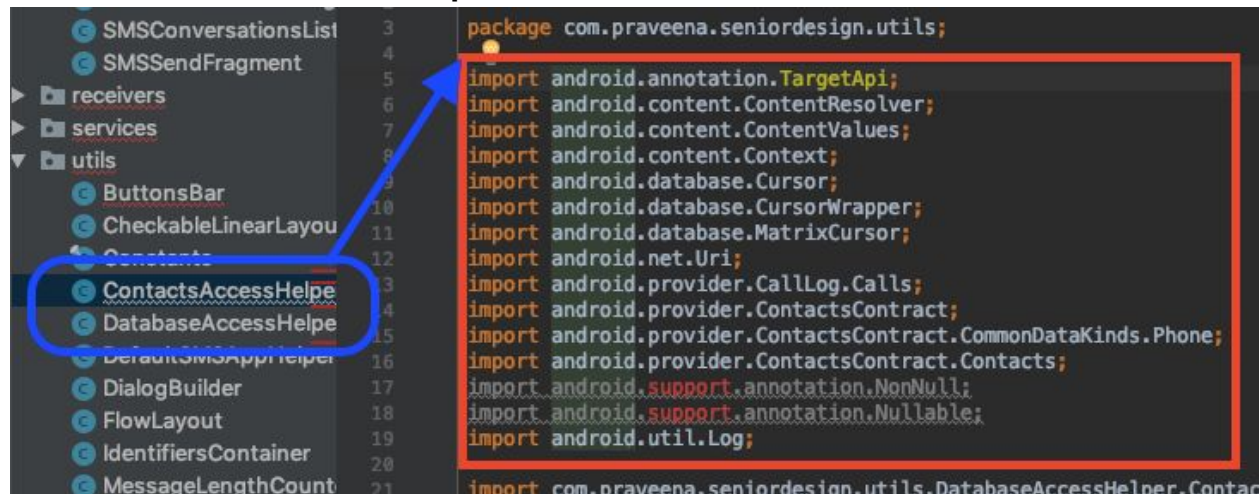
When initializing button, we have to be careful with the title that we use and make sure that the app listens to it whenever the user clicks it and if they can see the button or not.

### Checkable LinearLayout and Constants:

These help to align the app in a clean shape. The attributes that we have used here are `widget.Checkable`, `widget.LinearLayout` and other `attributeset`. This file helps to create the drawable state with a good enough space between text and buttons. *Constants* is the final java class which is setting the recent version API that we used:

```
public final class Constants {
    public static final int PIE_API_VERSION = 28;
}
```

### Contacts/Database Access Helper :



These files utilize the most packages compared to other files as we need to combine the backend and frontend to mix it and combine it. We have classified the number into five different types from here in order to access it easily.

```

// Types of contact sources
public enum ContactSourceType {
    FROM_CONTACTS,
    FROM_CALLS_LOG,
    FROM_SMS_LIST,
    FROM_BLACK_LIST,
    FROM_WHITE_LIST
}
  
```

This helped to achieve our main goal which was to block unwanted calls and in order to do so we had to find and classify each number based on the order above. After that, when the cursor points to getContacts, five different radio buttons will be displayed. In this case, when the user clicks the blacklist, database access helper will find all the numbers that he/she clicked in order to block the numbers and display it. The same process occurs for the whitelist.

### Dialog Builder:

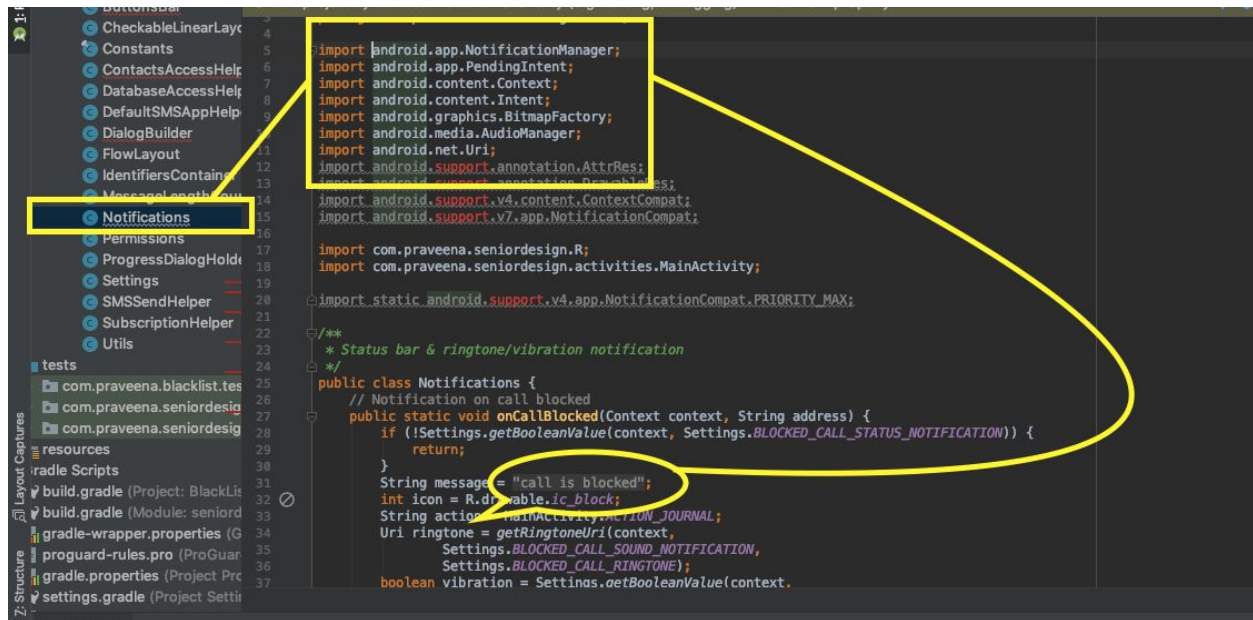
This file was created for the frontend part of the app which deals with the context, view, dialog, list layout and button bar. In order to set the title for each dialog box, we have to use the function called DialogBuilder which sets the linear layout and the title is fed by us. In order to set the visibility and add a new item to the list with the title only, we have used addItem where we return Id number and it will be produced.

### FlowLayout:

This is solely created in order to control the horizontal and vertical spacing of the app. Besides that, it controls the height and the width of the screen. It looks for left and right padding.



## Notifications:



The above figure shows the vivid information of how the notification is being handled in the app. As shown above, we have used notificationmanager, AudioManager for the tune of the message and Intent to access the contacts from android studio. The function Notification helps to set the ringtone/vibration notification and status bar. If the onCallBlocked is called, it automatically gives a notification of "call is blocked" and if the number is from the blacklist sms, it gives the notification "message is blocked".

## Settings:

```

public class Settings {
    public static final String BLOCK_CALLS_FROM_BLACK_LIST = "BLOCK_CALLS_FROM_BLACK_LIST";
    public static final String BLOCK_ALL_CALLS = "BLOCK_ALL_CALLS";
    public static final String BLOCK_CALLS_NOT_FROM_CONTACTS = "BLOCK_CALLS_NOT_FROM_CONTACTS";
    public static final String BLOCK_CALLS_NOT_FROM_SMS_CONTENT = "BLOCK_CALLS_NOT_FROM_SMS_CONTENT";
    public static final String BLOCK_PRIVATE_CALLS = "BLOCK_PRIVATE_CALLS";
    public static final String BLOCKED_CALL_STATUS_NOTIFICATION = "BLOCKED_CALL_STATUS_NOTIFICATION";
    public static final String WRITE_CALLS_JOURNAL = "WRITE_CALLS_JOURNAL";
    public static final String BLOCK_SMS_FROM_BLACK_LIST = "BLOCK_SMS_FROM_BLACK_LIST";
    public static final String BLOCK_ALL_SMS = "BLOCK_ALL_SMS";
    public static final String BLOCK_SMS_NOT_FROM_CONTACTS = "BLOCK_SMS_NOT_FROM_CONTACTS";
    public static final String BLOCK_SMS_NOT_FROM_SMS_CONTENT = "BLOCK_SMS_NOT_FROM_SMS_CONTENT";
    public static final String BLOCK_PRIVATE_SMS = "BLOCK_PRIVATE_SMS";
    public static final String BLOCKED_SMS_STATUS_NOTIFICATION = "BLOCKED_SMS_STATUS_NOTIFICATION";
    public static final String WRITE_SMS_JOURNAL = "WRITE_SMS_JOURNAL";
    public static final String BLOCKED_SMS_SOUND_NOTIFICATION = "BLOCKED_SMS_SOUND_NOTIFICATION";
    public static final String RECEIVED_SMS_SOUND_NOTIFICATION = "RECEIVED_SMS_SOUND_NOTIFICATION";
    public static final String BLOCKED_SMS_VIBRATION_NOTIFICATION = "BLOCKED_SMS_VIBRATION_NOTIFICATION";
    public static final String RECEIVED_SMS_VIBRATION_NOTIFICATION = "RECEIVED_SMS_VIBRATION_NOTIFICATION";
    public static final String BLOCKED_SMS_RINGTONE = "BLOCKED_SMS_RINGTONE";
    public static final String RECEIVED_SMS_RINGTONE = "RECEIVED_SMS_RINGTONE";
    public static final String BLOCKED_CALL_SOUND_NOTIFICATION = "BLOCKED_CALL_SOUND_NOTIFICATION";
    public static final String BLOCKED_CALL_VIBRATION_NOTIFICATION = "BLOCKED_CALL_VIBRATION_NOTIFICATION";
    public static final String BLOCKED_CALL_RINGTONE = "BLOCKED_CALL_RINGTONE";
    public static final String DELIVERY_SMS_NOTIFICATION = "DELIVERY_SMS_NOTIFICATION";
    public static final String FOLD_SMS_TEXT_IN_JOURNAL = "FOLD_SMS_TEXT_IN_JOURNAL";
    public static final String UI_THEME_DARK = "UI_THEME_DARK";
    public static final String GO_TO_JOURNAL_AT_START = "GO_TO_JOURNAL_AT_START";
    public static final String DEFAULT_SMS_APP_NATIVE_PACKAGE = "DEFAULT_SMS_APP_NATIVE_PACKAGE";
    public static final String DONT_EXIT_ON_BACK_PRESSED = "DONT_EXIT_ON_BACK_PRESSED";
    public static final String REMOVE_FROM_CALL_LOG = "REMOVE_FROM_CALL_LOG";
    public static final String SIM_SUBSCRIPTION_ID = "SIM_SUBSCRIPTION";
}

```

This can be found in one of the side bars of the app where the user can press the button “settings” and the above names that are shown are used in most of the java files but we had to rename them in a simpler format to track each variable.

```

public static void initDefaults(Context context) {
    Map<String, String> map = new HashMap<>();
    map.put(BLOCK_CALLS_FROM_BLACK_LIST, TRUE);
    map.put(BLOCK_ALL_CALLS, FALSE);
    map.put(BLOCK_CALLS_NOT_FROM_CONTACTS, FALSE);
    map.put(BLOCK_CALLS_NOT_FROM_SMS_CONTENT, FALSE);
    map.put(BLOCK_PRIVATE_CALLS, FALSE);
    map.put(WRITE_CALLS_JOURNAL, TRUE);
    map.put(BLOCKED_CALL_STATUS_NOTIFICATION, TRUE);
    map.put(BLOCKED_CALL_SOUND_NOTIFICATION, FALSE);
    map.put(BLOCKED_CALL_VIBRATION_NOTIFICATION, FALSE);
    map.put(BLOCK_SMS_FROM_BLACK_LIST, TRUE);
    map.put(BLOCK_ALL_SMS, FALSE);
    map.put(BLOCK_SMS_NOT_FROM_CONTACTS, FALSE);
    map.put(BLOCK_SMS_NOT_FROM_SMS_CONTENT, FALSE);
    map.put(BLOCK_PRIVATE_SMS, FALSE);
    map.put(BLOCKED_SMS_STATUS_NOTIFICATION, TRUE);
    map.put(WRITE_SMS_JOURNAL, TRUE);
    map.put(BLOCKED_SMS_SOUND_NOTIFICATION, FALSE);
    map.put(RECEIVED_SMS_SOUND_NOTIFICATION, TRUE);
    map.put(BLOCKED_SMS_SOUND_NOTIFICATION, FALSE);
    map.put(RECEIVED_SMS_VIBRATION_NOTIFICATION, TRUE);
    map.put(BLOCKED_SMS_VIBRATION_NOTIFICATION, FALSE);
    map.put(DELIVERY_SMS_NOTIFICATION, TRUE);
    map.put(FOLD_SMS_TEXT_IN_JOURNAL, TRUE);
    map.put(UI_THEME_DARK, FALSE);
    map.put(GO_TO_JOURNAL_AT_START, FALSE);
    map.put(DONT_EXIT_ON_BACK_PRESSED, FALSE);
    map.put(REMOVE_FROM_CALL_LOG, FALSE);
    map.put(SIM_SUBSCRIPTION_ID, v: "1");
}

```

The figure above shows the initial defaults that are set to boolean values for each variable. If the block\_calls\_from\_black\_list are set to true then it blocks the desired caller. Similarly, if the blocked\_Sms\_Sound\_notification is set to false then the mobile app will not produce any kind of sound and gets set to vibration automatically. However, we can change the settings according to the personal desire such as changing the ringtone, blocking the caller who are not sms texters or can change the whole outlook of the app to a dark or white theme. This gives freedom to the user to change the settings as they desire.

## Jupyter Notebook

---

Cell by cell view and explanation:

```
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import warnings
%matplotlib inline
```

At the beginning of the code, multiple libraries are imported to manipulate and access the provided datasets as well as to plot graphs. By importing *pandas*, a python library shown in *Figure 3*, it enables us to carry out entire data analysis workflow by accessing and manipulating data within the CSV files, in addition to being significantly useful in terms of Linear Regression models (*import LinearRegression*). In order to graph the given data values, *matplotlib.pyplot* is imported to provide interactive plots and simple cases for plot generation. Furthermore, by importing *statsmodels.api*, it allows us to create the tables of statistical data sourced from the CSV files. Three specific CSV data sets are downloaded, where the most recently updated file is the training dataset (*November 25*) and the other two datasets were used to test against it: the oldest CSV file provided (*October 21*) and the CSV file between the latest and oldest (*November 11*). ([Learn More](#))

```
train_df = pd.read_csv('dnc_complaint_numbers_2019-11-25.csv', index_col=None)
test_df = pd.read_csv('dnc_complaint_numbers_2019-10-21.csv', index_col=None)
submit1_df = pd.read_csv('dnc_complaint_numbers_2019-11-11.csv', index_col=None)
```

To run this code, begin by navigating to the Federal Trade Commission website and downloading the *dnc\_complaint\_numbers* csv files for November 25 2019, October 21 2019, and November 11 2019. These files contain reported certain phone numbers that were possible scam calls.



```

# Array using Consumer_States Provided
StateArray = ['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut', 'Dela
ware', 'Florida', 'Georgia',
              'Hawaii', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine'
, 'Maryland',
              'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi', 'Missouri', 'Montana', 'Nebraska'
, 'Nevada', 'New Hampshire', 'New Jersey',
              'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon', '
Pennsylvania', 'Rhode Island',
              'South Carolina', 'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Was
hington', 'West Virginia', 'Wisconsin', 'Wyoming',
              'Ontario, Canada', 'District of Columbia', 'Virgin Islands', 'Puerto Rico', 'Northern M
ariana Islands', 'US Military Pacific']

# Array corresponding Consumer State number
CorrStateNum = [1,2,3,4,5,6,7,8,9,10,
                11,12,13,14,15,16,17,18,19,20,
                21,22,23,24,25,26,27,28,29,30,
                31,32,33,34,35,36,37,38,39,40,
                41,42,43,44,45,46,47,48,49,50,
                51,52,53,54,55,56
                ]

# REPLACE Consumer_State Array with Corresponding Consumer_State Number Array
train_df = train_df.replace(StateArray, CorrStateNum)
test_df = test_df.replace(StateArray, CorrStateNum)
submit1_df = submit1_df.replace(StateArray, CorrStateNum)

modifiedtrain = train_df
modifiedtest = test_df
modifiedsubmit = submit1_df

```

The locations in the sourced data are all in strings and the code will run into a `TypeError`. Thus, each U.S. territory in the data has to be represented by specific integers. To do this, the locations are filtered alphabetically and each state is iteratively given a number.

```

# Replace scam call strings with integers: 1 -> Yes, 0 -> No
modifiedtrain = train_df['Recorded_Message_Or_Robocall'] = train_df['Recorded_Message_Or_Robocall'].replace('Y', 1)
modifiedtrain = train_df['Recorded_Message_Or_Robocall'] = train_df['Recorded_Message_Or_Robocall'].replace('N', 0)

modifiedtest = test_df['Recorded_Message_Or_Robocall'] = test_df['Recorded_Message_Or_Robocall'].replace('Y', 1)
modifiedtest = test_df['Recorded_Message_Or_Robocall'] = test_df['Recorded_Message_Or_Robocall'].replace('N', 0)

modifiedsubmit = submit1_df['Recorded_Message_Or_Robocall'] = submit1_df['Recorded_Message_Or_Robocall'].replace('Y', 1)
modifiedsubmit = submit1_df['Recorded_Message_Or_Robocall'] = submit1_df['Recorded_Message_Or_Robocall'].replace('N', 0)

#modifiedtrain['Company_Phone_Number'].sample(20)
# remove all null values
modifiedtrain = train_df.dropna()
modifiedtest = test_df.dropna()
modifiedsubmit = submit1_df.dropna()

modifiedtrain['Consumer_State'] = modifiedtrain['Consumer_State'].astype(int)
modifiedtest['Consumer_State'] = modifiedtest['Consumer_State'].astype(int)
modifiedsubmit['Consumer_State'] = modifiedsubmit['Consumer_State'].astype(int)

# Display 20 samples
modifiedtrain.sample(20)

```

The column *Recorded\_Message\_Or\_Robocall*, which determines whether the phone number was either a scam or not, indicated by Y or N, was converted to usable integer values of 1 or 0, respectively, by using the *replace* function provided by python. Furthermore, because certain values are NaN, *dropna* function was used to remove them, as well as update the changed values of *Consumer\_State* feature to integer types (*astype(int)*).

```

new_f = modifiedtest[['Company_Phone_Number']]

new_f.to_csv("OriginalScams.csv", index=False)

```

This code stores the first column dataset from our modified test file, *Company\_Phone\_Number*, into the variable *new\_f*, which is then stored within a CSV file created by using the function *to\_csv*.

```
# extract usable features
feature_cols = [
    'Recorded_Message_Or_Robocall',
    'Consumer_State'
]
train_features = modifiedtrain[feature_cols]

# construct target vector, dependent variable
train_target = modifiedtrain['Consumer_Area_Code']
```

A feature\_cols array is used to identify the features that are going to be used. The features in use are the binary variable “Recorded\_Message\_Or\_Robocall” and “Consumer\_State”. The target variable will be “Consumer\_Area\_Code”.

```
# using linear regression method
lreg = LinearRegression()
lreg.fit(train_features, train_target)
```

The model to be tested is Linear Regression and it needs to be fitted with the training features and the target.

```
test_features = modifiedtest[feature_cols]

# construct predictions
modifiedtest['predicted'] = lreg.predict(test_features)
print("Prediction using Linear Regression train data against Test1:")

print("Mean Squared Error for Test1 using Linear Regression:", mean_squared_error(modifiedtest['Consumer_Area_Code'], modifiedtest['predicted']))
```

After testing the dataset using Linear Regression, a mean squared error is calculated.

```
master_df = modifiedtrain.append(modifiedtest, sort=False)

master_features = master_df[feature_cols].fillna(master_df[feature_cols].median(), axis=0)
master_target = master_df['Consumer_Area_Code']
```

This code adds the modified test dataset to the end of the modified training dataset, and replaces any null values with the medians of the available values within the column (function *fillna* with *median()*).

```
#handling missing data
submit1_features = modifiedsubmit[feature_cols].fillna(master_df[feature_cols].median(), axis=0)
submit1_features.sample(20)
# 1 -> Yes, 0 -> No
```

This code *fillna* with *median()*, replaces any null values found within the *feature\_cols* with the median of the available values and displays a random sample amount of size 20.

```
modifiedsubmit['Company_Phone_Number'].sample(100)
```

This code displays a random dataset of size 100 retrieved from the *Company\_Phone\_Number* column of the modified submit data file.

```
# Write first column to csv file
new_f = master_df[['Company_Phone_Number']]

new_f.to_csv("ModifiedScams.csv", index=False)
```

This code stores the first column dataset from the newly generated scam numbers into the variable *new\_f*, which is then stored within a CSV file created by using the function *to\_csv*.

```
# Scatter plots of different features for test2 data:

modifiedsubmit.plot(kind='scatter',x='Recorded_Message_Or_Robocall',y='Consumer_Area_Code',color='red')
plt.show()

modifiedsubmit.plot(kind='scatter',x='Consumer_State',y='Consumer_Area_Code',color='blue')
plt.show()
```

The code generates a scatter plot graph, with the chosen features of *Consumer\_State* and *Recorded\_Message\_Or\_Robocall* against the target vector *Consumer\_Area\_Code*.



```
# Scatter plots of different features for training data:

master_df.plot(kind='scatter',x='Recorded_Message_Or_Robocall',y='Consumer_Area_Code',color='red'
)
plt.show()

master_df.plot(kind='scatter',x='Consumer_State',y='Consumer_Area_Code',color='blue')
plt.show()
```

The code generates a scatter plot graph, with the chosen features of *Consumer\_State* and *Recorded\_Message\_Or\_Robocall* against the target vector *Consumer\_Area\_Code*.

```
ax = plt.gca()
modifiedsubmit.plot(kind='line',x='Consumer_Area_Code',y='Recorded_Message_Or_Robocall',title='Co
rrelation of Recorded_Message_Or_Robocall and Consumer_State vs Consumer_Area_Code',ax=ax)
modifiedsubmit.plot(kind='line',x='Consumer_Area_Code',y='Consumer_State', color='red', ax=ax)
plt.show()
```

The two plots that were generated were scatters of the data with the x-axis being a binary digit, 1 or 0, and the y axis being the consumer area code.

```
#OLS Regression Results for Testing2 Data
# Add a constant to our existing dataframe for modeling purposes
modifiedsubmit = sm.add_constant(modifiedsubmit)

est = sm.OLS(modifiedsubmit['Consumer_Area_Code'],
              submit1_features[['Recorded_Message_Or_Robocall', 'Consumer_State']]
              ).fit()

print(est.summary())
```

This code prints out the OLS chart using the features that were previously specified. The OLS chart gives a general understanding of the modified submit test data results and correlations between the variables used and the target variable.

```
#OLS Regression Results for Training Data
# Add a constant to our existing dataframe for modeling purposes
master_df = sm.add_constant(master_df)

est = sm.OLS(master_df['Consumer_Area_Code'],
              master_features[['Recorded_Message_Or_Robocall', 'Consumer_State']]
              ).fit()

print(est.summary())
```

This code prints out the OLS chart using the features that were previously specified. This OLS chart gives a general understanding of the master\_df data results and correlations between the variables used. One of these metrics is the  $R^2$  value,



which tells us how correlated our feature variables are. The closer to 1 this metric is, the more correlated the variables are.

```
# Reading csv files -> OriginalScams.csv contains Company_Phone_Number sourced from FTC site
# Reading csv files -> ModifiedScams.csv contains Company_Phone_Number sourced from newly generated numbers
original = pd.read_csv('OriginalScams.csv')
newGen = pd.read_csv('ModifiedScams.csv')

'''
Test to see samples
-----
original['Company_Phone_Number'].sample(20)
newGen['Company_Phone_Number'].sample(20)
'''

# Create a dataframe to see if newGen values exists in original
dataframes = newGen.assign(Inoriginal=newGen.Company_Phone_Number.isin(original.Company_Phone_Number))

# Set values to see if InOriginal column is 1 or 0
numberInTrainingSet = dataframes[dataframes['Inoriginal'] == 1]
numberNotInTrainingSet = dataframes[dataframes['Inoriginal'] == 0]

# print the values of counted True's/ False's -> exists or does not exists
print("Total amount of numbers in dataset:\n", len(dataframes), end="")
print("Amount of numbers generated in original training set:\n", numberInTrainingSet.count())
print("Amount of numbers generated not in original training set:\n", numberNotInTrainingSet.count())

# Defining Variables
Total = len(dataframes)
InOriginalSet = numberInTrainingSet.count()
NotInOriginalSet = numberNotInTrainingSet.count()

# print the percentage values of Company_Phone_Numbers that exists in the original from our newly generated values
print("Percentage in original training set: ", (InOriginalSet / Total) * 100)
print("Percentage not in original training set: ", (NotInOriginalSet / Total) * 100)
```

To calculate the accuracy of the newly generated data, the function `pd.read_csv` opens the CSV files that previously had the written data (*OriginalScam.csv* and *ModifiedScams.csv*) and created a dataframe that generated a new column *Inoriginal*, which returns T or F, by checking each value in the *ModifiedScams.csv* dataset against the old FTC dataset in *OriginalScams.csv*. To calculate how much new data exists within the old, two variables *numberInTrainingSet* and *numberNotInTrainingSet* are initialized to store data values of 1 or 0 (T or F), followed by the python `count` function. After acquiring the total count of existing and non-existing values, as well as the total number of values within the dataset (datasets in both CSV files are of equal length), the equations (displayed as the last two lines in the snippet above) print out the accuracy.