Near-Earth-Objects: Are We Doomed?

An analysis by Jacqueline Tsodikova, Alejandra Magana, Robert Janke, Michael Albers, Fred Jambor

What are we analyzing?

Our group is researching the likelihood an asteroid or comet will potentially harm our planet. Using historical data, we will determine what thresholds are used to predict which NEOs are the most hazardous to Earth.

Why this topic?

A new movie on Netflix, *Don't Look Up*, was just released that had to do with a comet approaching Earth and scientists trying to warn the public about it. Although this movie is more about comedic humor, a comet or asteroid harming our planet is definitely something that could happen. We wanted to research and analyze the data to see how likely a NEO could harm us.

How many close approach objects will we have in the next decade?

On average, 200-400 space objects enter Earth's atmosphere every year, that's about one a day. But, these are usually small and not hazardous

Can we predict potentially hazardous objects in the future?

The probability of an object to be PHA is small, but the damages are definitely big. A comet that was 10 kilometers is what wiped out the dinosaurs.

Which NEOs are the most potentially hazardous?

Knowing exactly which NEOs are the most hazardous can help researchers track those more closely.

Technologise Used:

- Tableau for our dashboard
- AWS to store data
- Pandas for cleaning the tables
- Random Forest Classifier to test our overall performance
- PostgreSQL to store large and sophisticated data safely
- PySpark to have a wide range of libraries
- Entity Relationship DIagram (ERD) to model the stored data

Description of Our Data Source

Our data comes from the Center For Near Earth Object Studies (CENOS) a NASA center that computes highly accurate orbital data for thousands of asteroids and comets in our solar system.

CNEOS collects it's information from minor planet center which includes orbital data and physical properties like size and rotation rates.

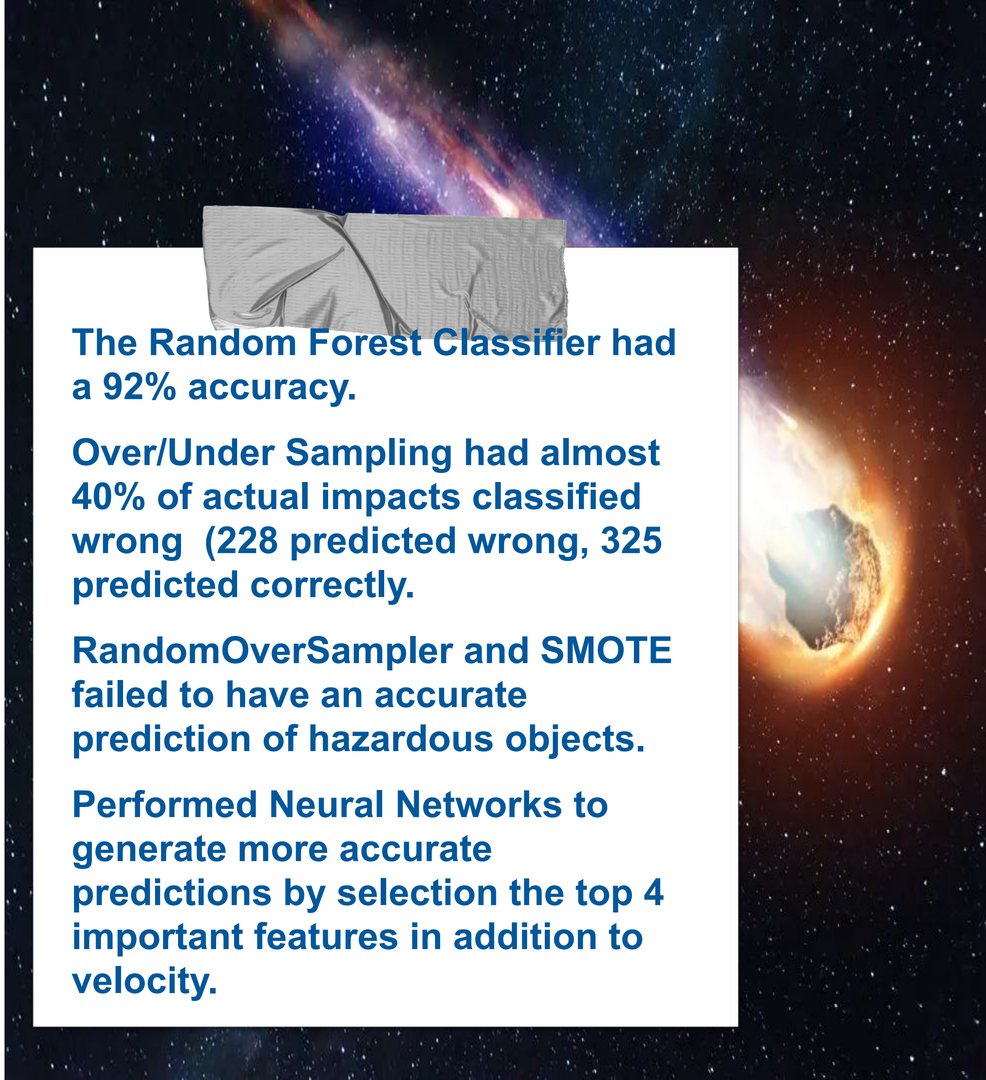
Description of the Data Exploration Phase

Our database consisted of 29,052 rows and 36 columns.

Dropped string columns containing names, IDs, equinox and PC because they had no impact to our analysis.

Eliminated the columns with null values more than 50% of the total number of rows and replaced the other null values with 0.

Description of the Analysis Phase



The Random Forest Classifier had a 92% accuracy.

Over/Under Sampling had almost 40% of actual impacts classified wrong (228 predicted wrong, 325 predicted correctly.

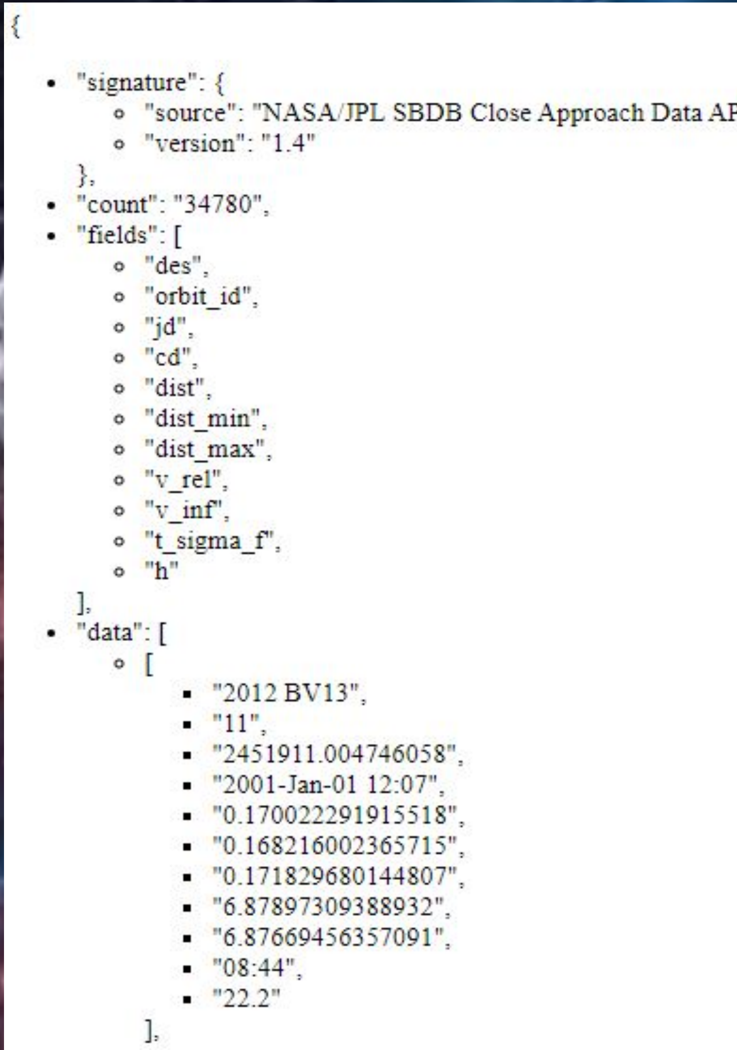
RandomOverSampler and SMOTE failed to have an accurate prediction of hazardous objects.

Performed Neural Networks to generate more accurate predictions by selection the top 4 important features in addition to velocity.

For our database, since we had over 28,000 rows, we decided to use PostgreSQL because it stores large and sophisticated data safely and we could visually see the relationships between our data.

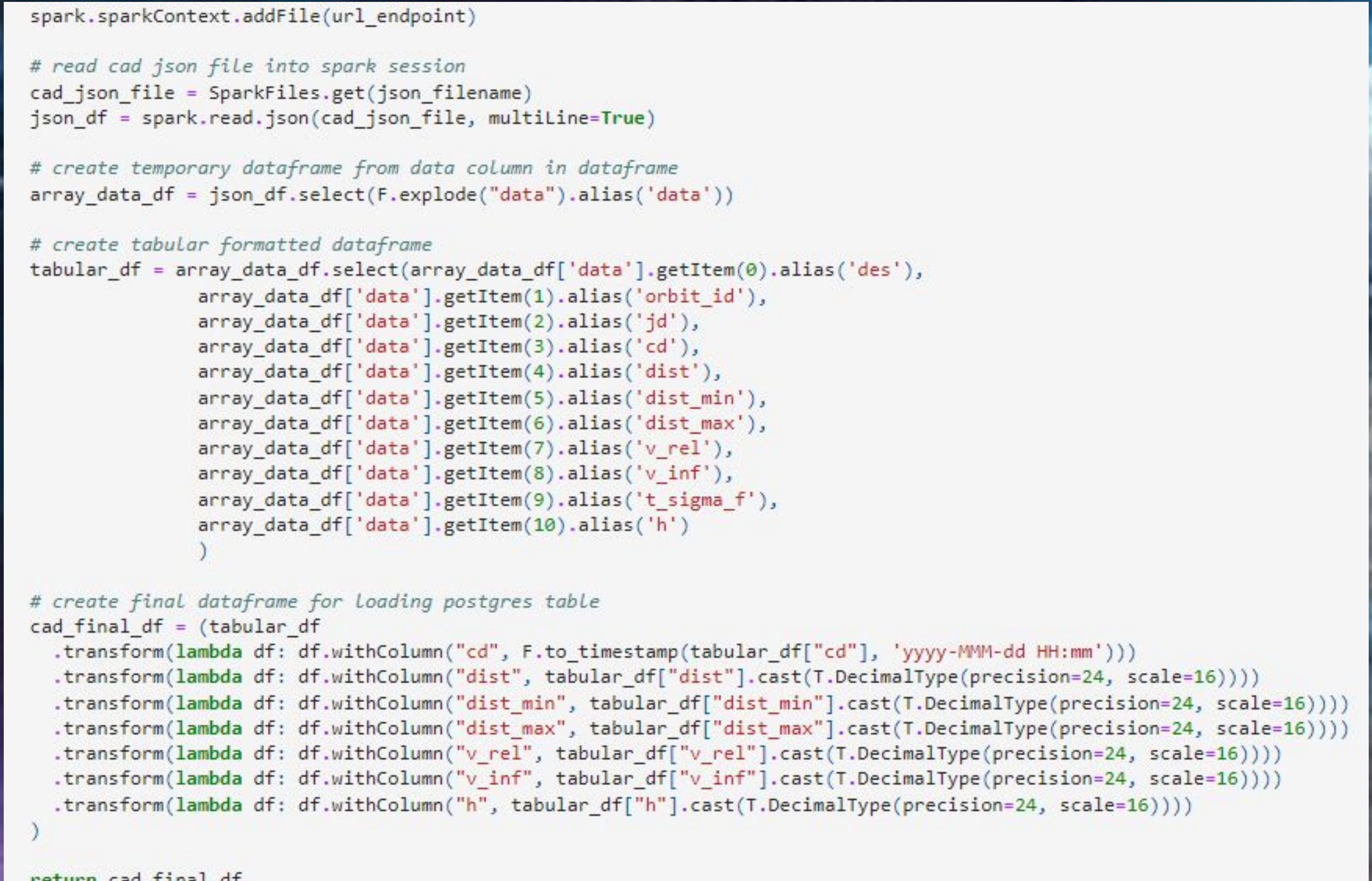
Database

ETL Process: JSON Data



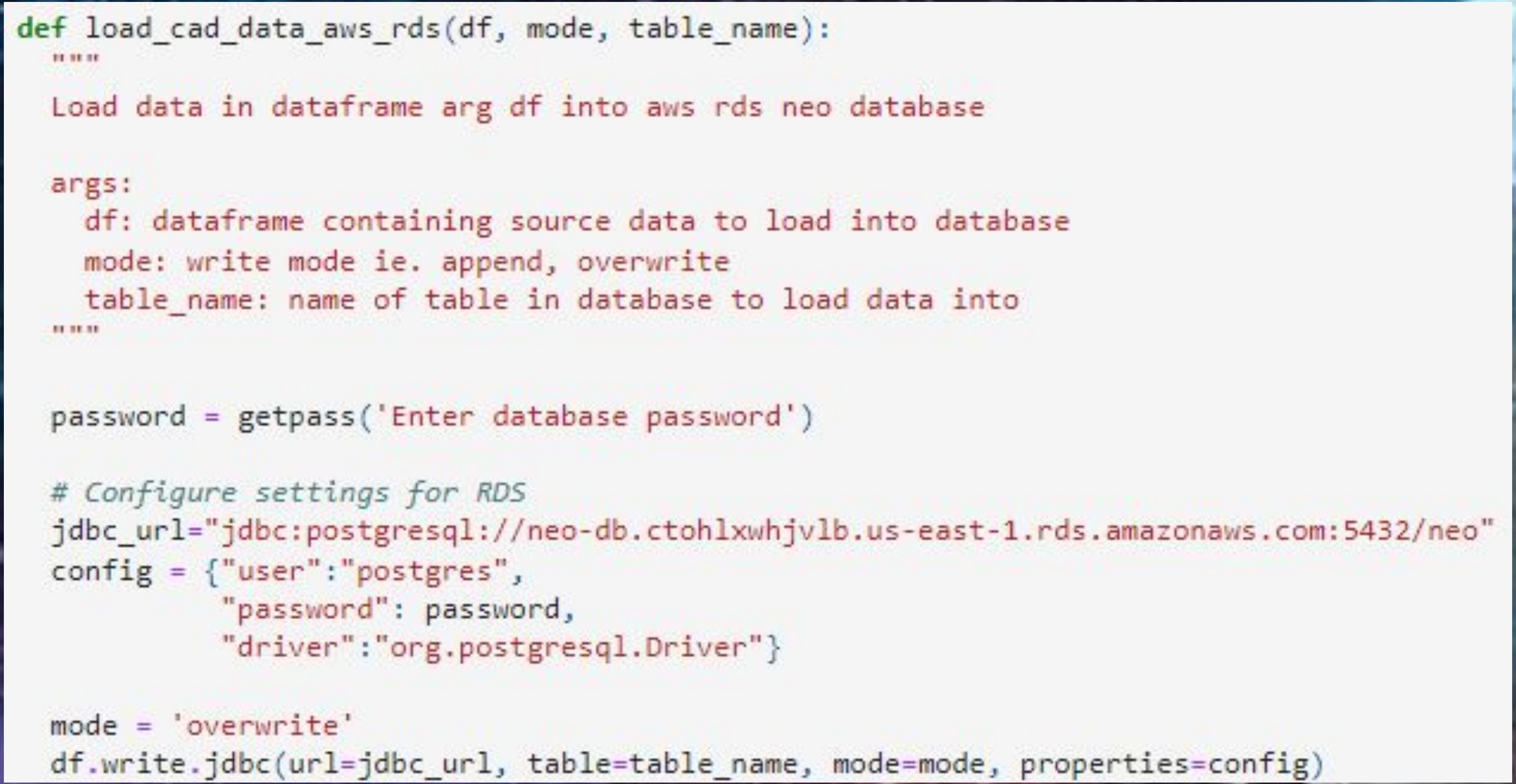
```
{
  • "signature": {
      ○ "source": "NASA/JPL SBDB Close Approach Data AP
      ○ "version": "1.4"
    },
  • "count": "34780",
  • "fields": [
      ○ "des",
      ○ "orbit_id",
      ○ "jd",
      ○ "cd",
      ○ "dist",
      ○ "dist_min",
      ○ "dist_max",
      ○ "v_rel",
      ○ "v_inf",
      ○ "t_sigma_f",
      ○ "h"
    ],
  • "data": [
      ○ [
          ▪ "2012 BV13",
          ▪ "11",
          ▪ "2451911.004746058",
          ▪ "2001-Jan-01 12:07",
          ▪ "0.170022291915518",
          ▪ "0.168216002365715",
          ▪ "0.171829680144807",
          ▪ "6.87897309388932",
          ▪ "6.87669456357091",
          ▪ "08:44",
          ▪ "22.2"
        ],
```

Transformation



```python
spark.sparkContext.addFile(url_endpoint)

# read cad json file into spark session
cad_json_file = SparkFiles.get(json_filename)
json_df = spark.read.json(cad_json_file, multiLine=True)

# create temporary dataframe from data column in dataframe
array_data_df = json_df.select(F.explode("data").alias('data'))

# create tabular formatted dataframe
tabular_df = array_data_df.select(array_data_df['data'].getItem(0).alias('des'),
               array_data_df['data'].getItem(1).alias('orbit_id'),
               array_data_df['data'].getItem(2).alias('jd'),
               array_data_df['data'].getItem(3).alias('cd'),
               array_data_df['data'].getItem(4).alias('dist'),
               array_data_df['data'].getItem(5).alias('dist_min'),
               array_data_df['data'].getItem(6).alias('dist_max'),
               array_data_df['data'].getItem(7).alias('v_rel'),
               array_data_df['data'].getItem(8).alias('v_inf'),
               array_data_df['data'].getItem(9).alias('t_sigma_f'),
               array_data_df['data'].getItem(10).alias('h')
               )

# create final dataframe for loading postgres table
cad_final_df = (tabular_df
  .transform(lambda df: df.withColumn("cd", F.to_timestamp(tabular_df["cd"], 'yyyy-MMM-dd HH:mm')))
  .transform(lambda df: df.withColumn("dist", tabular_df["dist"].cast(T.DecimalType(precision=24, scale=16))))
  .transform(lambda df: df.withColumn("dist_min", tabular_df["dist_min"].cast(T.DecimalType(precision=24, scale=16))))
  .transform(lambda df: df.withColumn("dist_max", tabular_df["dist_max"].cast(T.DecimalType(precision=24, scale=16))))
  .transform(lambda df: df.withColumn("v_rel", tabular_df["v_rel"].cast(T.DecimalType(precision=24, scale=16))))
  .transform(lambda df: df.withColumn("v_inf", tabular_df["v_inf"].cast(T.DecimalType(precision=24, scale=16))))
  .transform(lambda df: df.withColumn("h", tabular_df["h"].cast(T.DecimalType(precision=24, scale=16))))
)

return cad_final_df
```

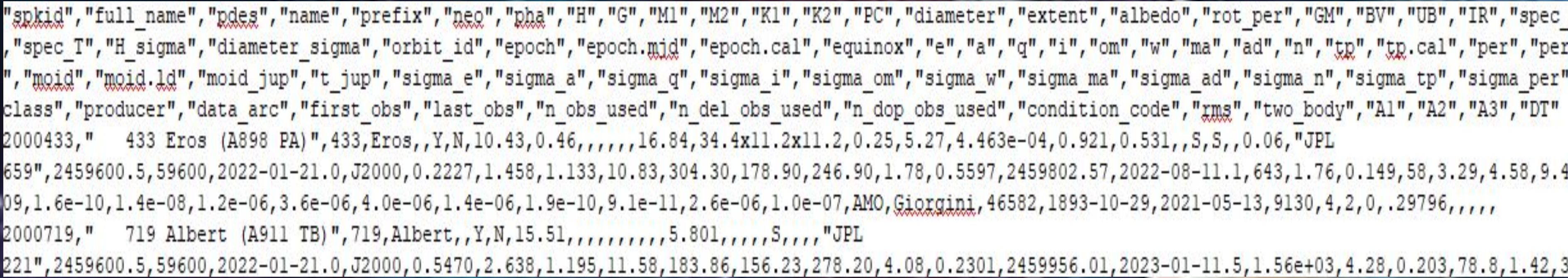
Load



```
def load_cad_data_aws_rds(df, mode, table_name):
  """
  Load data in dataframe arg df into aws rds neo database

  args:
    df: dataframe containing source data to load into database
    mode: write mode ie. append, overwrite
    table_name: name of table in database to load data into
  """

  password = getpass('Enter database password')

  # Configure settings for RDS
  jdbc_url="jdbc:postgresql://neo-db.ctohlxwhjvlb.us-east-1.rds.amazonaws.com:5432/neo"
  config = {"user":"postgres",
            "password": password,
            "driver":"org.postgresql.Driver"}

  mode = 'overwrite'
  df.write.jdbc(url=jdbc_url, table=table_name, mode=mode, properties=config)
```

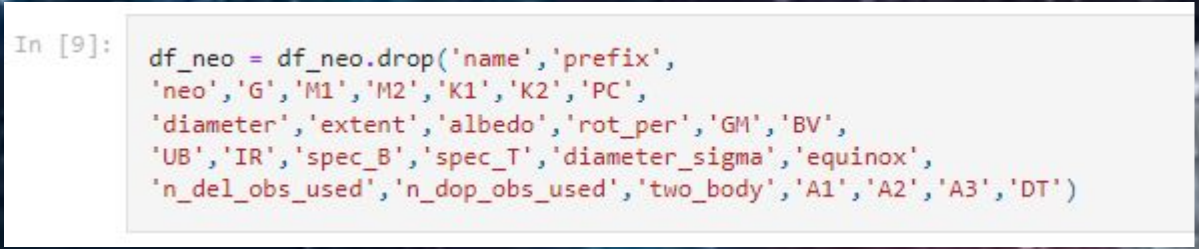
ETL Process: NEOs

1. Download CSV

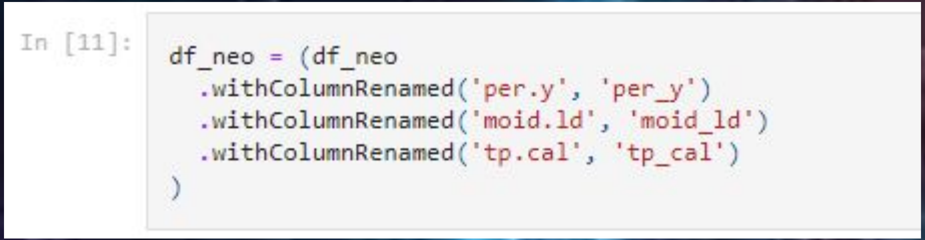


```
"spkid","full_name","pdes","name","prefix","neo","pha","H","G","M1","M2","K1","K2","PC","diameter","extent","albedo","rot_per","GM","BV","UB","IR","spec_
,"spec_T","H_sigma","diameter_sigma","orbit_id","epoch","epoch.mjd","epoch.cal","equinox","e","a","q","i","om","w","ma","ad","n","tp","tp.cal","per","per
","moid","moid.ld","moid_jup","t_jup","sigma_e","sigma_a","sigma_q","sigma_i","sigma_om","sigma_w","sigma_ma","sigma_ad","sigma_n","sigma_tp","sigma_per"
class","producer","data_arc","first_obs","last_obs","n_obs_used","n_del_obs_used","n_dop_obs_used","condition_code","rms","two_body","A1","A2","A3","DT"
2000433,"   433 Eros (A898 PA)",433,Eros,,Y,N,10.43,0.46,,,,,,16.84,34.4x11.2x11.2,0.25,5.27,4.463e-04,0.921,0.531,,S,S,,0.06,"JPL
659",2459600.5,59600,2022-01-21.0,J2000,0.2227,1.458,1.133,10.83,304.30,178.90,246.90,1.78,0.5597,2459802.57,2022-08-11.1,643,1.76,0.149,58,3.29,4.58,9.4
09,1.6e-10,1.4e-08,1.2e-06,3.6e-06,4.0e-06,1.4e-06,1.9e-10,9.1e-11,2.6e-06,1.0e-07,AMO,Giorgini,46582,1893-10-29,2021-05-13,9130,4,2,0,.29796,,,,,
2000719,"   719 Albert (A911 TB)",719,Albert,,Y,N,15.51,,,,,,,,,,5.801,,,,,S,,,,"JPL
221",2459600.5,59600,2022-01-21.0,J2000,0.5470,2.638,1.195,11.58,183.86,156.23,278.20,4.08,0.2301,2459956.01,2023-01-11.5,1.56e+03,4.28,0.203,78.8,1.42,3
```

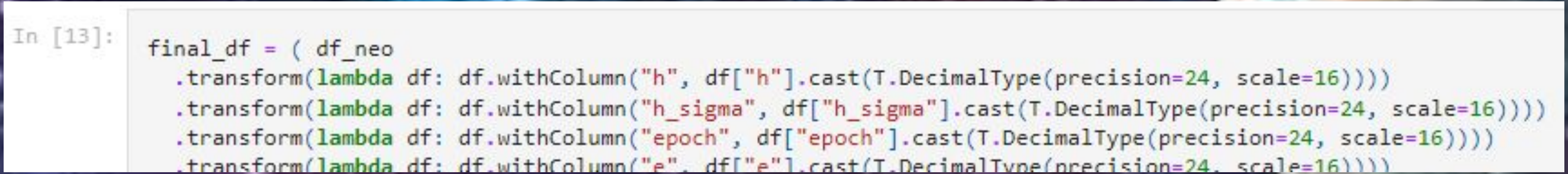
2. Transformation



```
In [9]:  df_neo = df_neo.drop('name','prefix',
         'neo','G','M1','M2','K1','K2','PC',
         'diameter','extent','albedo','rot_per','GM','BV',
         'UB','IR','spec_B','spec_T','diameter_sigma','equinox',
         'n_del_obs_used','n_dop_obs_used','two_body','A1','A2','A3','DT')
```

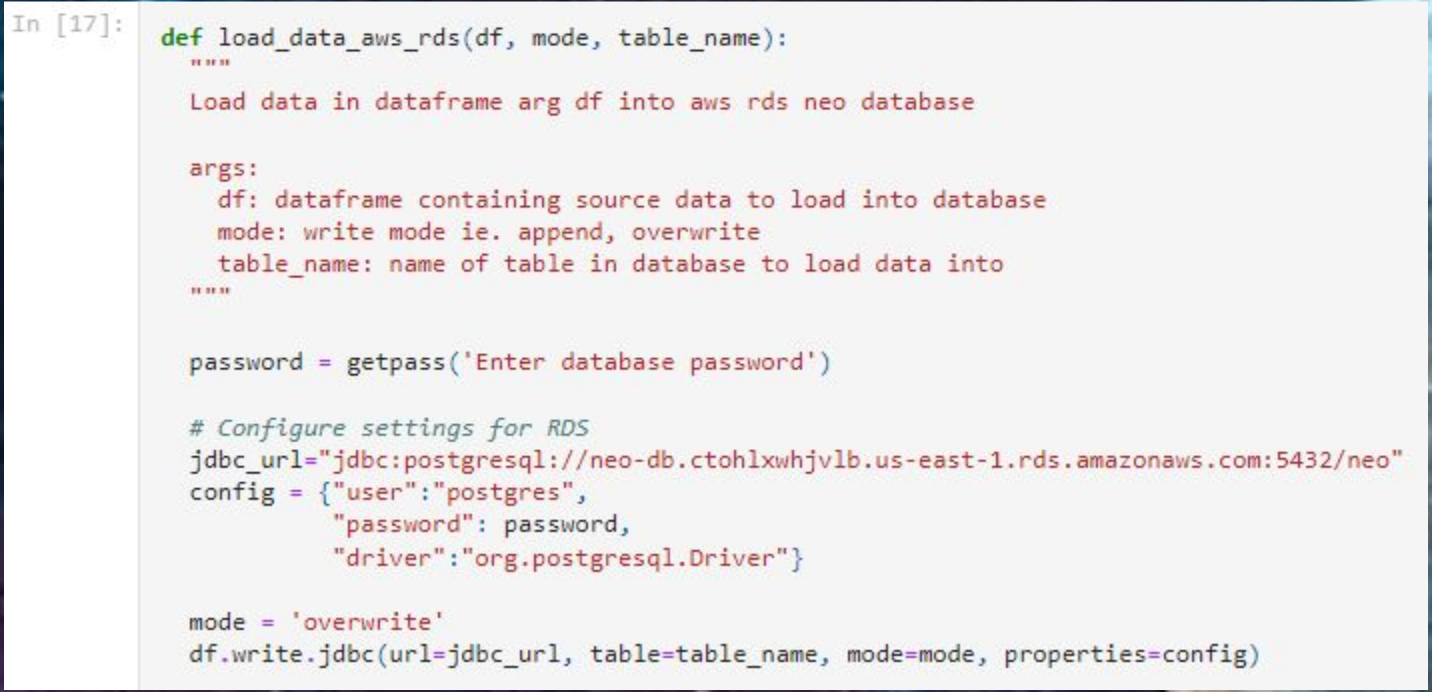


```
In [11]: df_neo = (df_neo
           .withColumnRenamed('per.y', 'per_y')
           .withColumnRenamed('moid.ld', 'moid_ld')
           .withColumnRenamed('tp.cal', 'tp_cal')
         )
```



```
In [13]: final_df = ( df_neo
           .transform(lambda df: df.withColumn("h", df["h"].cast(T.DecimalType(precision=24, scale=16))))
           .transform(lambda df: df.withColumn("h_sigma", df["h_sigma"].cast(T.DecimalType(precision=24, scale=16))))
           .transform(lambda df: df.withColumn("epoch", df["epoch"].cast(T.DecimalType(precision=24, scale=16))))
           .transform(lambda df: df.withColumn("e", df["e"].cast(T.DecimalType(precision=24, scale=16))))
```

3. Load



```
In [17]:
def load_data_aws_rds(df, mode, table_name):
  """
  Load data in dataframe arg df into aws rds neo database

  args:
    df: dataframe containing source data to load into database
    mode: write mode ie. append, overwrite
    table_name: name of table in database to load data into
  """

  password = getpass('Enter database password')

  # Configure settings for RDS
  jdbc_url="jdbc:postgresql://neo-db.ctohlxwhjvlb.us-east-1.rds.amazonaws.com:5432/neo"
  config = {"user":"postgres",
            "password": password,
            "driver":"org.postgresql.Driver"}

  mode = 'overwrite'
  df.write.jdbc(url=jdbc_url, table=table_name, mode=mode, properties=config)
```

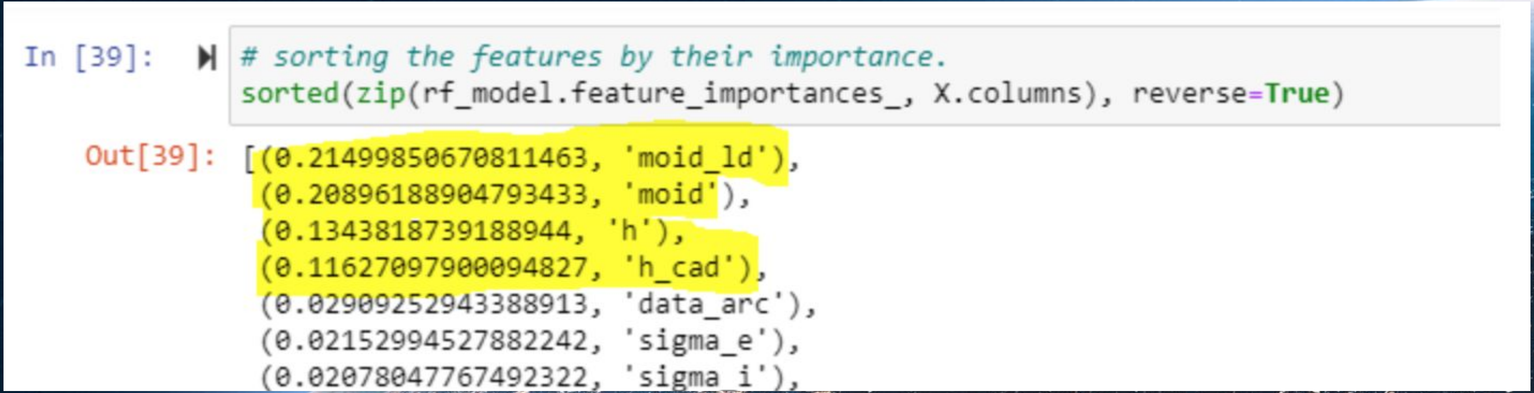
Machine Learning

Performing Resampling

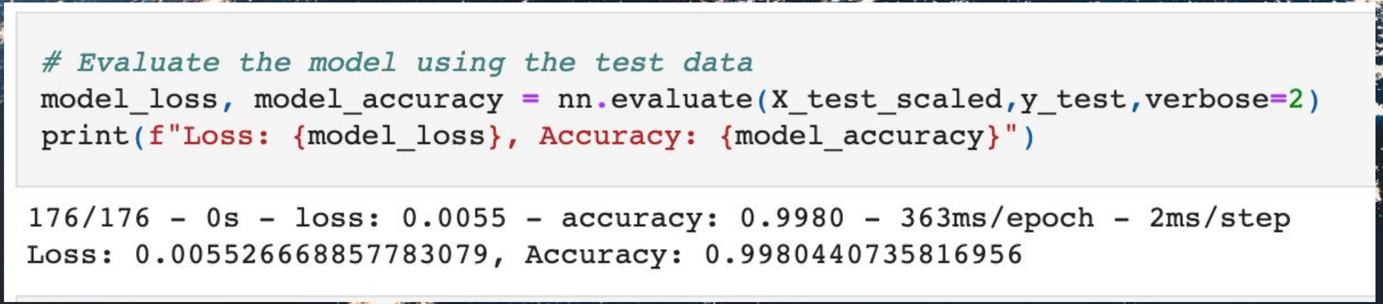
- Random Forest
- Random - Over - Sampler
- Over and Under Sampling for DS2
- SMOTE
- Neural Networks

Choosing Our Model

Neural Networks: 99.8% accuracy
However, true negatives make up a lot of that percentage, so our confidence matrix will show that most of the predictions were correct



```
In [39]:    # sorting the features by their importance.
            sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)

Out[39]:  [(0.21499850670811463, 'moid_ld'),
           (0.20896188904793433, 'moid'),
           (0.1343818739188944, 'h'),
           (0.11627097900094827, 'h_cad'),
           (0.02909252943388913, 'data_arc'),
           (0.02152994527882242, 'sigma_e'),
           (0.02078047767492322, 'sigma_i'),
```



```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

176/176 - 0s - loss: 0.0055 - accuracy: 0.9980 - 363ms/epoch - 2ms/step
Loss: 0.005526668857783079, Accuracy: 0.9980440735816956
```

Clustering

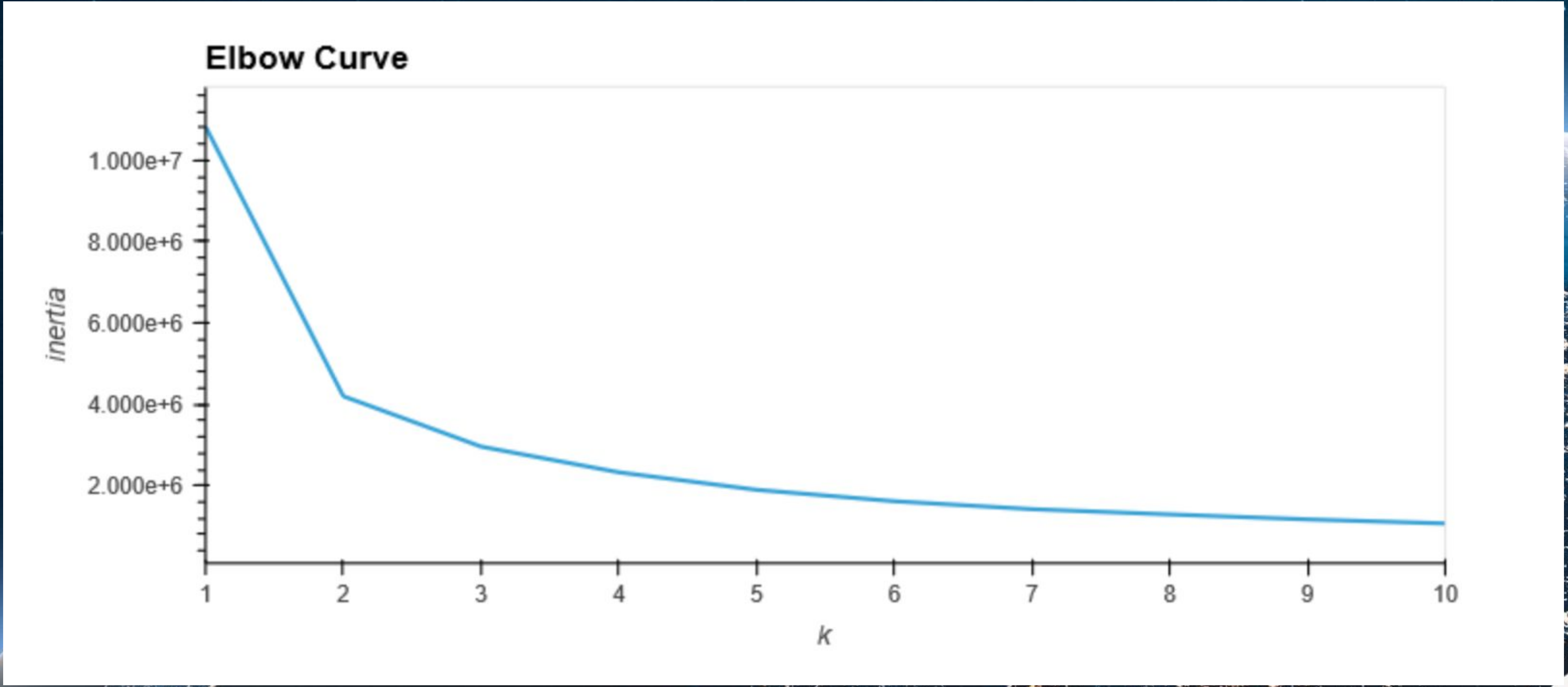
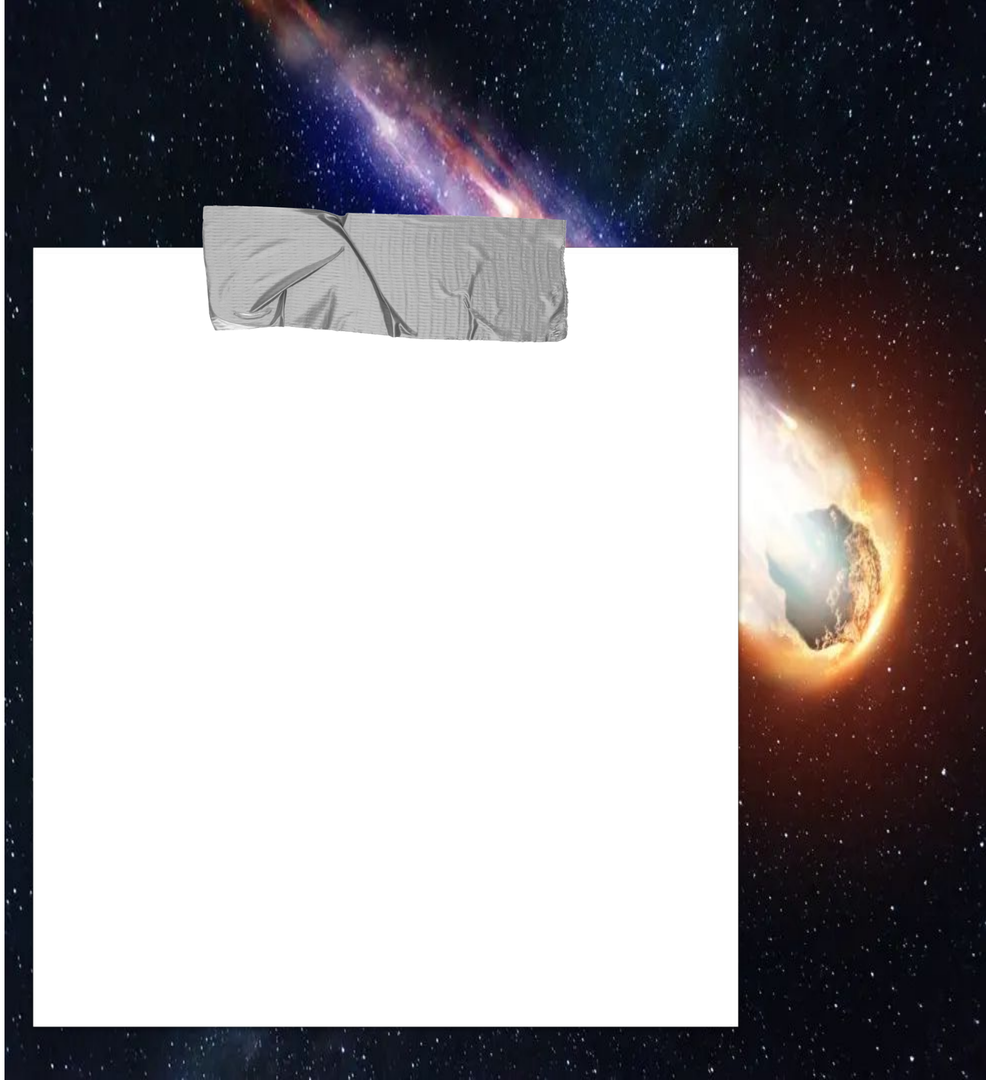

Elbow Curve
inertia
1.000e+7
8.000e+6
6.000e+6
4.000e+6
2.000e+6
1
2
3
4
5
6
7
8
9
10
k

Tableau Story

-

Description of the Interactive Elements



Analysis Results