DEW POINT AFFECTED PROBABILITY

Data Processing and Feature Engineering:

1) I encoded the categorical variables into integer values using the Label Encoder. Here's the code:

```
[23] #colums_to_convert are all the columns that have object type
     columns_to_convert = ['THROW_SIDE_KEY', 'PITCH_TYPE_TRACKED_KEY', 'EVENT_RESULT_KEY', 'PITCH_RESULT_KEY']
     label_encoder = LabelEncoder()
     for col in columns_to_convert:
         data[col] = label_encoder.fit_transform(data[col])
```

Determining the significance of features in relation to the output we aim to predict, which is the dew probability:

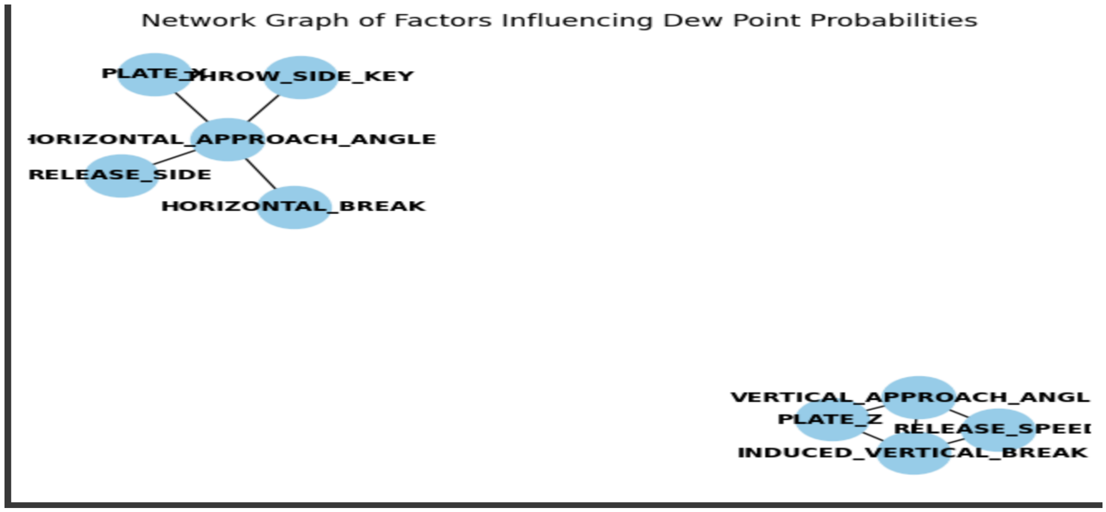
1)I began by investigating the key factors influencing dew point probability. Initial analysis using chi-square tests revealed a significant correlation between two categorical variables: 'pitch_result_key' and 'event_result_key.'

2)Considering the influence of atmospheric conditions on 'induced vertical break' and 'horizontal break,' I proceeded to analyze the correlations between all continuous variables using both Pearson's and Spearman's coefficients. I found some strong correlation between some variables.

```
Strong Positive Correlations:
                   Variable 1                  Variable 2  \
42        BATTER_IN_INNING_KEY                    OUT_KEY
61                PITCH_NUMBER                      BALLS
62                PITCH_NUMBER                    STRIKES
178     INDUCED_VERTICAL_BREAK              RELEASE_SPEED
183     INDUCED_VERTICAL_BREAK    VERTICAL_APPROACH_ANGLE
207              RELEASE_SPEED    VERTICAL_APPROACH_ANGLE
212               RELEASE_SIDE  HORIZONTAL_APPROACH_ANGLE
216             RELEASE_HEIGHT          RELEASE_EXTENSION
226  HORIZONTAL_APPROACH_ANGLE                    PLATE_X
229    VERTICAL_APPROACH_ANGLE                    PLATE_Z

     Pearson's Correlation  Spearman's Correlation
42                0.678483                0.765956
61                0.809877                0.815230
62                0.790129                0.824165
178               0.646535                0.668569
183               0.781252                0.780925
207               0.685221                0.691106
212              -0.726819               -0.650606
216              -0.512797               -0.429141
226               0.573896                0.565477
229               0.779972                0.779839
Strong Negative Correlations:
Empty DataFrame
Columns: [Variable 1, Variable 2, Pearson's Correlation, Spearman's Correlation]
Index: []
```

3)To prepare for training, I used a correlation map or heatmap to finalize the attributes. During this process, I identified a set of variables that exhibit the strongest correlations with each other.



Methodology:

1)To prepare for modeling, I first organized the dataset by grouping it based on 'inning_key' and 'pitcher_key,' capitalizing on the assumption that pitcher behavior remains consistent within innings.
2)Within each of these individual groups, I utilized K-means Clustering to pinpoint anomalies, helping to identify any unusual patterns or outliers in the data.
3)Subsequently, I developed an Autoencoder model, where non-anomalous data within each group served as the input.
4)During the testing phase, I applied the Autoencoder model to the entire dataset, evaluating its performance across the board.
5)To estimate the likelihood of anomalies, I computed reconstruction errors, enabling the calculation of the probabilities for each pitch data point being affected by dew levels greater than 65 degrees.

```
[54] # Using Autoencoder model to do anomaly detection.
     # the bigger reconstruction error, the bigger will be the probability of it being an anomaly.
     # error is directly proportional to probability of it being affected by dew point greater than 65
     class Autoencoder(nn.Module):
         def __init__(self, input_dim):
             super(Autoencoder, self).__init__()
             self.encoder = nn.Sequential(
                 nn.Linear(input_dim, 64),
                 nn.ReLU(),
                 nn.Linear(64, 32),
                 nn.ReLU(),
                 nn.Linear(32, 16),
                 nn.ReLU(),
                 nn.Linear(16, 8)
             )
             self.decoder = nn.Sequential(
                 nn.Linear(8, 16),
                 nn.ReLU(),
                 nn.Linear(16, 32),
                 nn.ReLU(),
                 nn.Linear(32, 64),
                 nn.ReLU(),
                 nn.Linear(64, input_dim)
             )

         def forward(self, x):
             x = self.encoder(x)
             x = self.decoder(x)
             return x
```