final_notebook

December 1, 2020

[1]:
```
# Standard Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Models
from sklearn.linear_model import LinearRegression, PoissonRegressor
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# Modeling Evaluation
from sklearn.preprocessing import StandardScaler, LabelEncoder, LabelBinarizer,␣
↪PolynomialFeatures, OneHotEncoder
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

import os, sys

sys.path.append("../../src")
from helper_functions import MAPE, plot_feature_importances,␣
↪plot_top5_feature_importances, plot_obs_pred

%load_ext autoreload
%autoreload 2
```

[3]:
```
sns.set(context = 'notebook', style = 'whitegrid')
ppt_colors = ['#9146ff', '#571dad', '#9168cd', '#bd9eeb', '#b58eee', '#b48bf0']
```

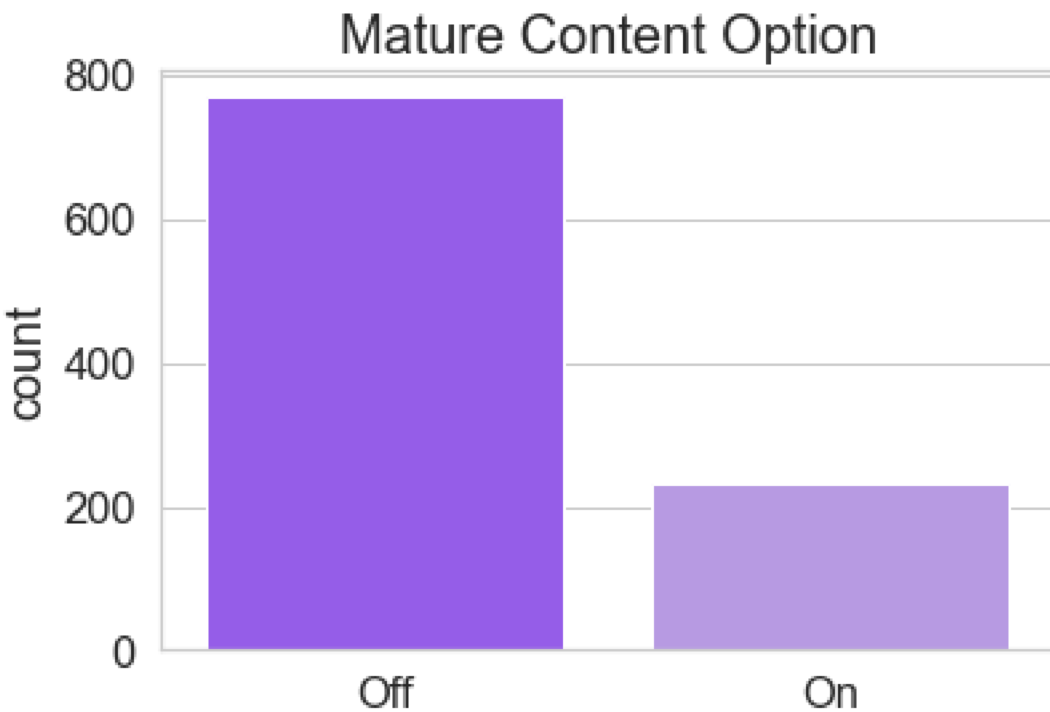
[4]:
```
# Data should be placed in the data/raw folder in order to run the code

raw_data_path = os.path.join(os.pardir, os.pardir, "data", "raw",␣
↪"twitchdata-update.csv")
df_twitch = pd.read_csv(raw_data_path)
```

[5]:
```
# Let's look at the data

df_twitch.head(5)
```

[5]:
```
    Channel  Watch time(Minutes)  Stream time(minutes)  Peak viewers  \
0     xQcOW           6196161750                215250        222720
1  summit1g           6091677300                211845        310998
2    Gaules           5644590915                515280        387315
3  ESL_CSGO           3970318140                517740        300575
4      Tfue           3671000070                123660        285644

   Average viewers  Followers  Followers gained  Views gained  Partnered  \
0            27716    3246298           1734810      93036735       True
1            25610    5310163           1370184      89705964       True
2            10976    1767635           1023779     102611607       True
3             7714    3944850            703986     106546942       True
4            29602    8938903           2068424      78998587       True

   Mature    Language
0   False     English
1   False     English
2    True  Portuguese
3   False     English
4   False     English
```

1 Initial EDA

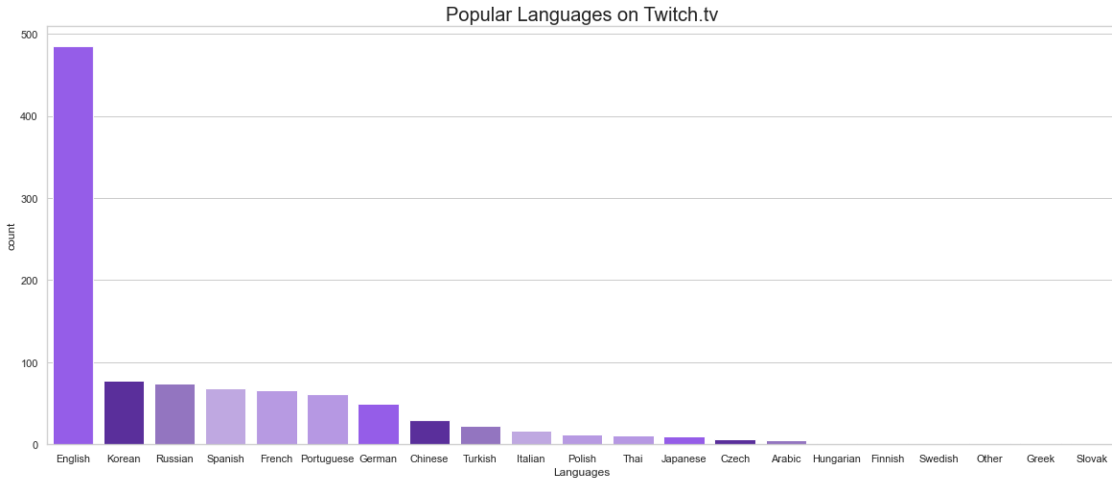
[6]:
```
# Mature content or Not

sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)

ax = sns.countplot(x = 'Mature', data = df_twitch, palette = [ppt_colors[0],␣
 ↪ppt_colors[4]])
ax.set(xlabel = '', xticklabels = ['Off', 'On'])
ax.set_title('Mature Content Option', fontdict={'fontsize': 20})

#plt.savefig('../../reports/figures/Mature.png', dpi = 500, bbox_inches =␣
 ↪'tight', transparent = True)
plt.show()
```



[7]:
```
# Partnered or Not

sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)

ax = sns.countplot(x = 'Partnered', data = df_twitch, palette = [ppt_colors[0],␣
↪ppt_colors[1]], order = df_twitch['Partnered'].value_counts().index)
ax.set(xlabel = '', xticklabels = ['True', 'False'])
ax.set_title('Twitch Partners', fontdict={'fontsize': 20})

#plt.savefig('../../reports/figures/Partnered.png', dpi = 500, bbox_inches =␣
↪'tight', transparent = True)
plt.show()
```

[8]:

```
# Language breakdown
plt.figure(figsize=(20, 8))
sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1)
xlabel = 'Languages'
ylabel = 'count'

ax = sns.countplot(x = 'Language', data = df_twitch, order =␣
 ↪df_twitch['Language'].value_counts().index, palette = ppt_colors)
ax.set(xlabel = xlabel, ylabel = ylabel)
ax.set_title('Popular Languages on Twitch.tv', fontdict={'fontsize': 20})

#plt.savefig('../../reports/figures/Languages.png', dpi = 500, bbox_inches =␣
 ↪'tight', transparent = True)
plt.show()
```



[9]:
```python
# Label Encoding Partnered column True : 1, False : 0

Partnered_le = df_twitch.Partnered

le = LabelEncoder()
le.fit(Partnered_le)
df_twitch.Partnered = le.transform(Partnered_le)
```

[10]:
```python
# Label Encoding Mature column True : 1, False : 0

Mature_le = df_twitch.Mature

le = LabelEncoder()
le.fit(Mature_le)
df_twitch.Mature = le.transform(Mature_le)
```

[11]:
```python
# Label Encoding Language column, numbers assigned to languages by default.

Language_le = df_twitch.Language

le = LabelEncoder()
le.fit(Language_le)
df_twitch.Language = le.transform(Language_le)
```

[12]:
```python
# Set a channel name as an index

df_twitch.set_index('Channel', drop=True, inplace=True)
df_twitch.head()
```

[12]:

```
             Watch time(Minutes)  Stream time(minutes)  Peak viewers  \
Channel
xQcOW                 6196161750                215250        222720
summit1g              6091677300                211845        310998
Gaules                5644590915                515280        387315
ESL_CSGO              3970318140                517740        300575
Tfue                  3671000070                123660        285644

             Average viewers  Followers  Followers gained  Views gained  \
Channel
xQcOW                  27716    3246298           1734810      93036735
summit1g               25610    5310163           1370184      89705964
Gaules                 10976    1767635           1023779     102611607
ESL_CSGO                7714    3944850            703986     106546942
Tfue                   29602    8938903           2068424      78998587

             Partnered  Mature  Language
Channel
xQcOW                1       0         3
summit1g             1       0         3
Gaules               1       1        14
ESL_CSGO             1       0         3
Tfue                 1       0         3
```

[13]:

```python
# Dropping channels that didn't gain followers
df_twitch_drop = df_twitch[df_twitch['Followers gained'] < 0].index
df_twitch_drop
```

[13]:

```
Index(['Amaz', 'TSM_TheOddOne', '   (newmasca)'], dtype='object',
name='Channel')
```

[14]:

```python
df_twitch = df_twitch.drop(df_twitch_drop)
```

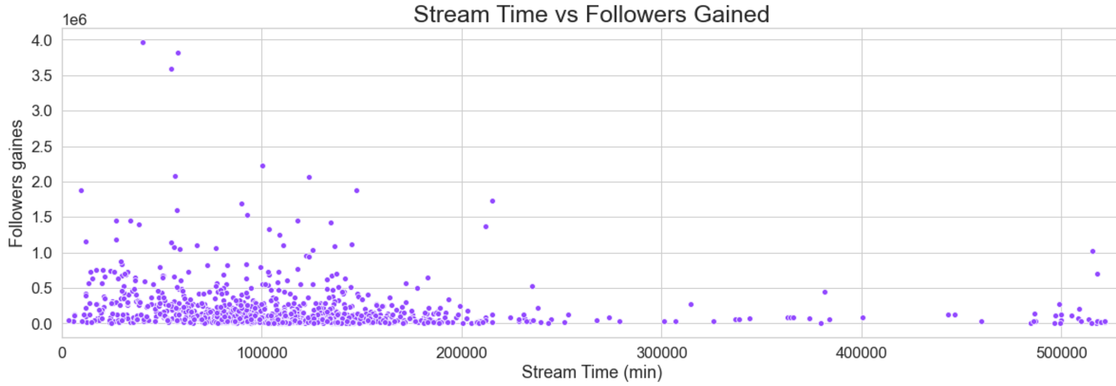
[15]:

```python
# clean data

df_twitch.to_csv('../../data/clean/CleanData.csv')
```

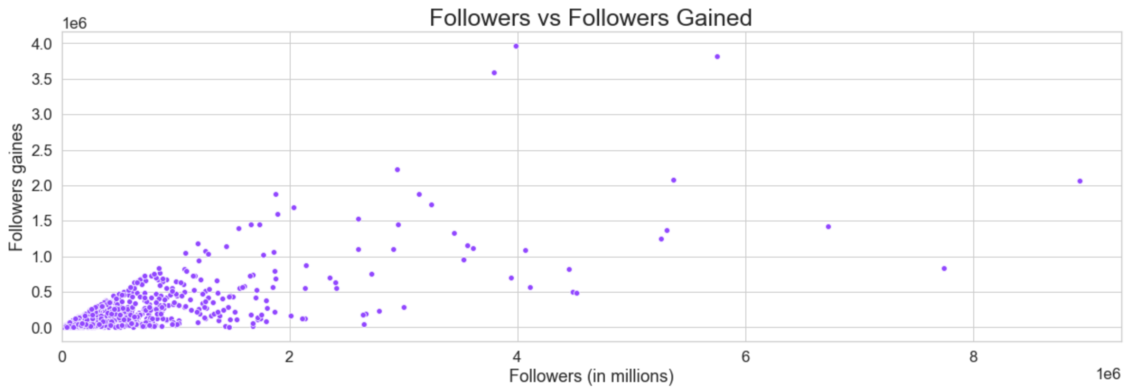
[16]:

```python
# Scatter plot
plt.figure(figsize=(20, 6))
sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)

line = sns.scatterplot(x='Stream time(minutes)', y='Followers gained',␣
 ↪data=df_twitch,  color = "#9146ff")
line.set(xlabel='Stream Time (min)', ylabel='Followers gaines')
line.set_title('Stream Time vs Followers Gained', fontdict={'fontsize': 25})
plt.xlim(left=0, right=5.3e5)
```

```
#plt.savefig("../../reports/figures/StreamTimevsFollowersgained.png",␣
↪bbox_inches='tight', dpi=500)
```



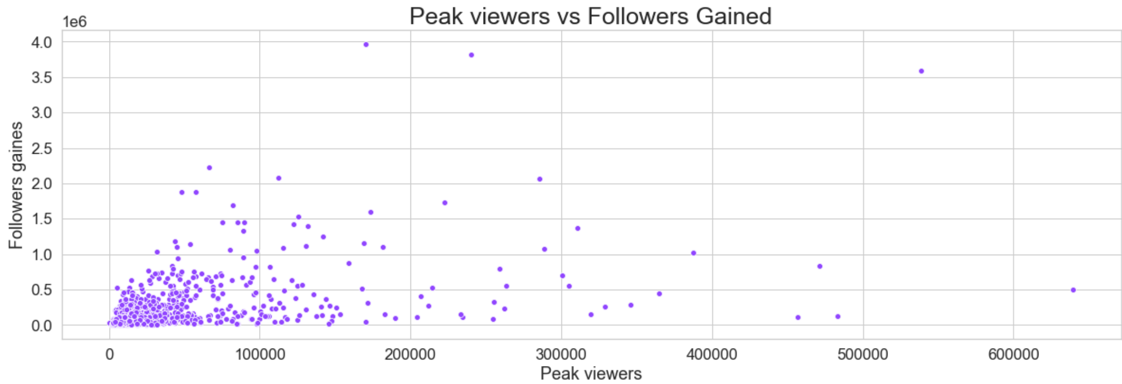
[17]:
```
# Scatter plot
plt.figure(figsize=(20, 6))
sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)

line = sns.scatterplot(x='Followers', y='Followers gained', data=df_twitch, ␣
↪color = "#9146ff")
line.set(xlabel='Followers (in millions)', ylabel='Followers gaines')
line.set_title('Followers vs Followers Gained', fontdict={'fontsize': 25})
plt.xlim(left=0, right=9.3e6)

#plt.savefig("../../reports/figures/FollowersvsFollowersgained.png",␣
↪bbox_inches='tight', dpi=500)
```



[18]:
```
# Scatter plot
plt.figure(figsize=(20, 6))
sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)
```

```
line = sns.scatterplot(x='Peak viewers', y='Followers gained', data=df_twitch, ␣
↪color = "#9146ff")
line.set(xlabel='Peak viewers', ylabel='Followers gaines')
line.set_title('Peak viewers vs Followers Gained', fontdict={'fontsize': 25})
```

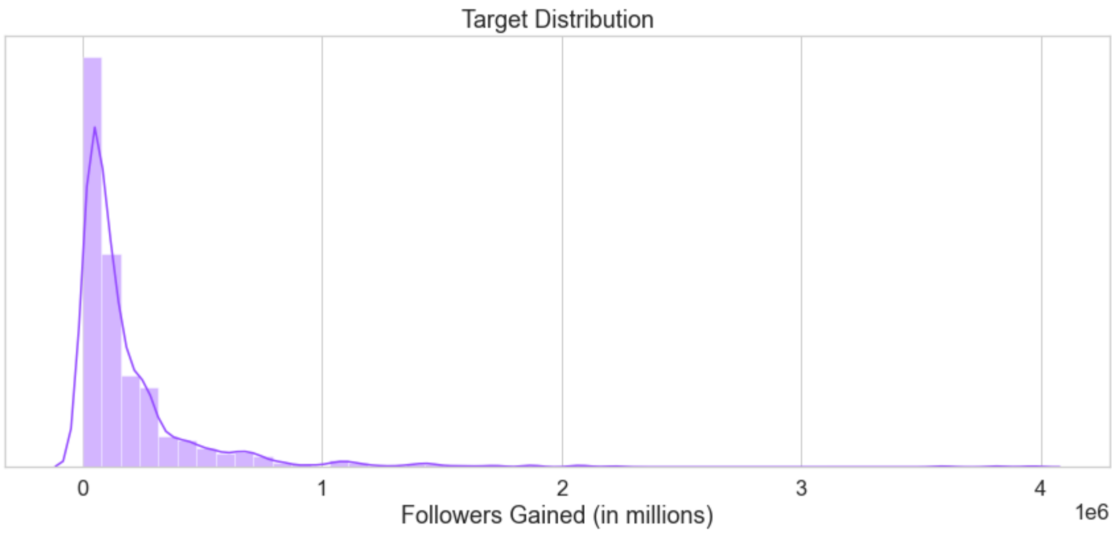
[18]: Text(0.5, 1.0, 'Peak viewers vs Followers Gained')



2 Data Preprocessing

[19]:
```
# Separating target y from features X

X = df_twitch.drop(['Followers gained'], axis=1)
y = df_twitch['Followers gained']
```

[20]:
```
# Plot Target Distribution

plt.figure(figsize=(15, 6))
sns.set(context = 'notebook', style = 'whitegrid', font_scale = 1.5)

target_dist = sns.distplot(y, color = "#9146ff")
target_dist.set(xlabel='Followers Gained (in millions)', title='Target␣
↪Distribution', yticks=[])

#plt.savefig("../../reports/figures/Target_Dist.png", bbox_inches='tight',␣
↪dpi=500)
```



[21]:
```
# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,␣
↪test_size=0.25)
```

[22]:
```
# Creating data frames for plotting final model performances

df_train = X_train.merge(y_train, on='Channel')
df_test = X_test.merge(y_test, on='Channel')
```

[23]:
```
# Standard Scaling for Poisson Regressor
ss = StandardScaler()
ss.fit(X_train)
X_t_ss = ss.transform(X_train)
X_test_ss = ss.transform(X_test)
```

3 FSM: Linear Regression

Linear Regression Model as First Simple Model

[24]:
```
# Create linear regression object
fsm = LinearRegression()

# Train the model using the training set
fsm.fit(X_train, y_train)
```

[24]:
```
LinearRegression()
```

[25]:
```
# Make predictions using the training set
y_pred_lr = fsm.predict(X_train)
```

[27]:
```
# Cross-validation on the training set
scores = cross_val_score(fsm, X_train, y_train, cv=5).mean()
scores
```

[27]:
```
0.4700925657247339
```

FSM scores for the test data

[26]:
```
# Make predictions using the testing set
y_pred_lr = fsm.predict(X_test)
```

[28]:
```
# The mean absolut percentage error
print('Mean absolute percentage error: %.2f'
      % MAPE(y_test, y_pred_lr))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %3f'
      % r2_score(y_test, y_pred_lr))
```

```
Mean absolute percentage error: 156.25
Coefficient of determination: 0.384980
```

3.0.1 Random Forest

Let's run Random Forest and hypertune parameters using Grid Search method

[29]:
```
rfr = RandomForestRegressor()
```

[30]:
```
# Define the parameter grid
# n_estimators - number of trees in the forest
# max_features - max number of features considered for splitting a node
# max_depth - max number of levels in each decision tree
# min_samples_leaf - min number of data points allowed in a leaf node

grid = {
    'n_estimators': [100, 1000],
    'max_features': ["auto", "sqrt", "log2"],
    'max_depth': [None, 3, 5, 10, 25],
    'min_samples_leaf': [1, 5, 10]

}
```

[31]:
```
# Initialize the gridsearch object with three-fold cross-validation
gs = GridSearchCV(rfr, grid, cv=3)
```

[32]:
```
gs.fit(X_train, y_train)
```

[32]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [None, 3, 5, 10, 25],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 5, 10],
                         'n_estimators': [100, 1000]})
```

[33]:
```
gs.best_params_
```

[33]:
```
{'max_depth': 5,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'n_estimators': 100}
```

[34]:
```
gs.best_score_
```

[34]:
```
0.5001601011878541
```

[35]:
```
# Final Random Forest model with best parameters
rfr_final = RandomForestRegressor(n_estimators=gs.best_params_['n_estimators'],␣
 ↪min_samples_leaf=gs.best_params_['min_samples_leaf'], random_state=42,␣
 ↪max_features=gs.best_params_['max_features'], max_depth=gs.
 ↪best_params_['max_depth'])
```

[36]:
```
rfr_final.fit(X_train, y_train)
```

[36]:
```
RandomForestRegressor(max_depth=5, random_state=42)
```

[37]:
```
y_pred_rf = rfr_final.predict(X_train)
```

[38]:
```
# Cross-validation on the training set
scores = cross_val_score(rfr, X_train, y_train, cv=5)
scores
```

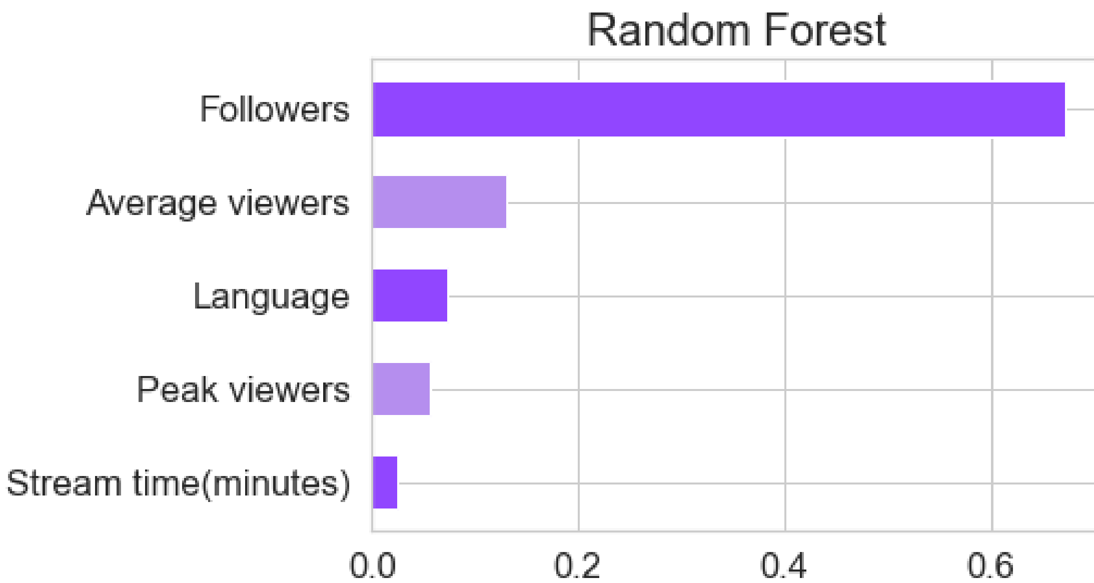
[38]:
```
array([0.48646376, 0.57910006, 0.4285909 , 0.7480087 , 0.52046788])
```

[39]:
```
y_pred_rf = rfr_final.predict(X_test)
```

[40]:
```
print('Mean absolute percentage error: %.2f'
      % MAPE(y_test, y_pred_rf))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %2f'
      % r2_score(y_test, y_pred_rf))
```

```
Mean absolute percentage error: 129.31
Coefficient of determination: 0.601475
```

[74]:
```
plot_top5_feature_importances(rfr_final, X_test, 'Random Forest')

plt.savefig("../../reports/figures/RF_5_feature_importances.png",␣
 ↪bbox_inches='tight', dpi=400)
```



3.0.2 Poisson Regressor

The data falls on Poisson distribution and meet all the requirements. The target is a random number of a gained followers over given period of time.

[42]:
```
pr = PoissonRegressor(max_iter=1000)
```

[43]:
```
pr.fit(X_t_ss, y_train)
```

[43]:
```
PoissonRegressor(max_iter=1000)
```

[44]:
```
# Cross-validation
scores = cross_val_score(pr, X_t_ss, y_train, cv=5)
scores
```

[44]:
```
array([ 0.34131693,  0.42767562,  0.37369452, -0.15536657,  0.6991641 ])
```

[45]:
```
y_pred_pr = pr.predict(X_test_ss)
```

[48]:
```
print('Mean absolute percentage error: %.2f'
      % MAPE(y_test, y_pred_pr))
# The coefficient of determination: 1 is perfect prediction
```

```
print('Coefficient of determination: %2f'
      % r2_score(y_test, y_pred_pr))
```

```
Mean absolute percentage error: 219.30
Coefficient of determination: 0.214750
```

3.0.3 K-Nearest Neighbors Regressor

Let's try KNN, it should perform well for a small data set.

[49]:
```
knn = KNeighborsRegressor()
```

[50]:
```
# Define the parameter grid
grid_knn = {
    'n_neighbors': [5, 10, 25],
    'weights': ["uniform", "distance"],
    'leaf_size': [1, 5, 10]

}
```

[51]:
```
# Initialize the gridsearch object with three-fold cross-validation
gs_knn = GridSearchCV(knn, grid_knn, cv=3)
```

[52]:
```
gs_knn.fit(X_train, y_train)
```

[52]:
```
GridSearchCV(cv=3, estimator=KNeighborsRegressor(),
             param_grid={'leaf_size': [1, 5, 10], 'n_neighbors': [5, 10, 25],
                         'weights': ['uniform', 'distance']})
```

[53]:
```
gs_knn.best_params_
```

[53]:
```
{'leaf_size': 1, 'n_neighbors': 25, 'weights': 'uniform'}
```

[54]:
```
gs_knn.best_score_
```

[54]:
```
0.16005701347499623
```

[55]:
```
# Cross-validation
scores = cross_val_score(gs_knn, X_train, y_train, cv=5)
scores
```

[55]:
```
array([ 0.25638666,  0.21369715, -0.08447832,  0.32962982,  0.22372753])
```

[56]:
```
y_pred_knn = gs_knn.predict(X_test)
```

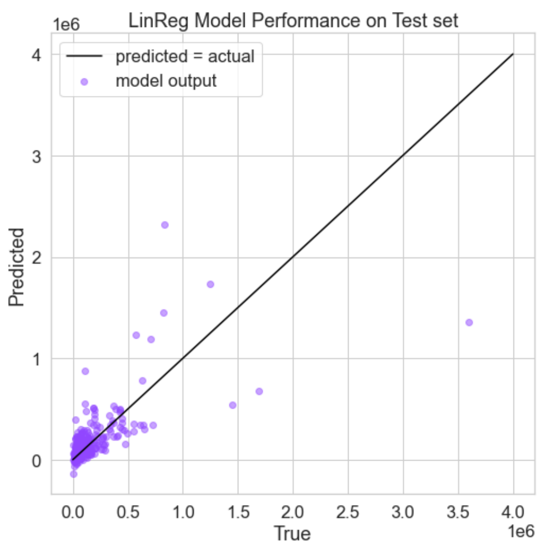
[58]:
```
print('Mean absolute percentage error: %.2f'
      % MAPE(y_test, y_pred_knn))
# The coefficient of determination: 1 is perfect prediction
```

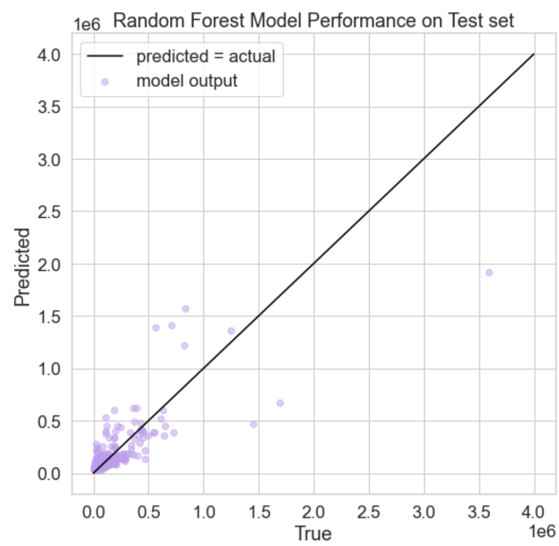
```
print('Coefficient of determination: %2f'
      % r2_score(y_test, y_pred_knn))
```

```
Mean absolute percentage error: 287.80
Coefficient of determination: 0.187928
```

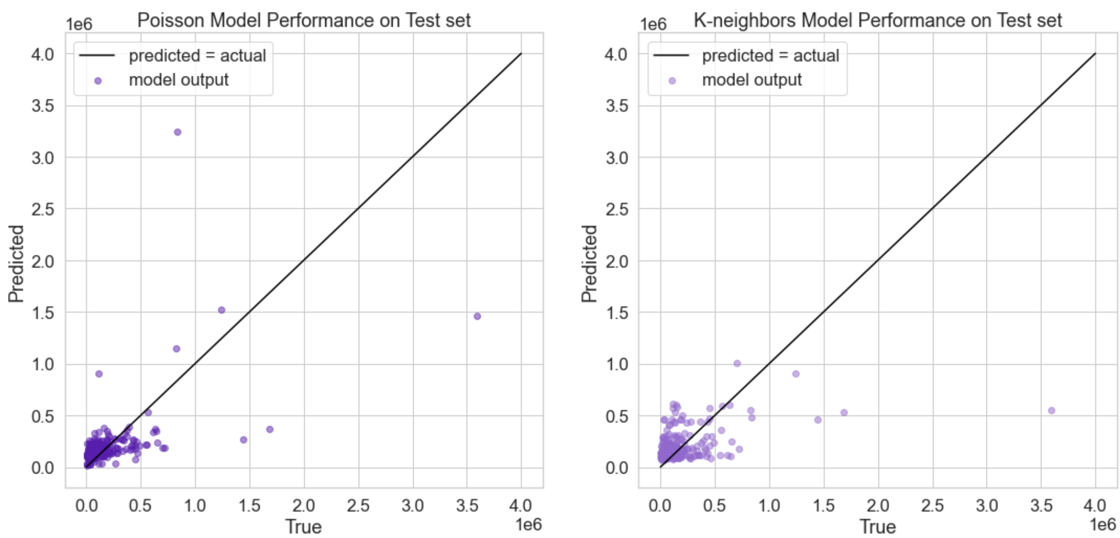
3.0.4 Plotting Final Results

[72]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 8))

# Customize which model is actually being plotted
ax1.set_title("LinReg Model Performance on Test set")
ax1.scatter(y_test, fsm.predict(X_test),
                alpha=0.5, label="model output", color=ppt_colors[0])

ax2.set_title("Random Forest Model Performance on Test set")
ax2.scatter(y_test, rfr_final.predict(X_test),
                alpha=0.5, label="model output", color=ppt_colors[3])

# Same setup for both plots (x and y labels, line showing y=x)
y_equals_x = np.linspace(0, 4e6)
for ax in (ax1, ax2):
    ax.set_xlabel("True")
    ax.set_ylabel("Predicted")
    ax.plot(y_equals_x, y_equals_x, label="predicted = actual", color="black")
    ax.legend()

plt.savefig("../../reports/figures/Models_performances_1.png",␣
 ↪bbox_inches='tight', dpi=400)
```





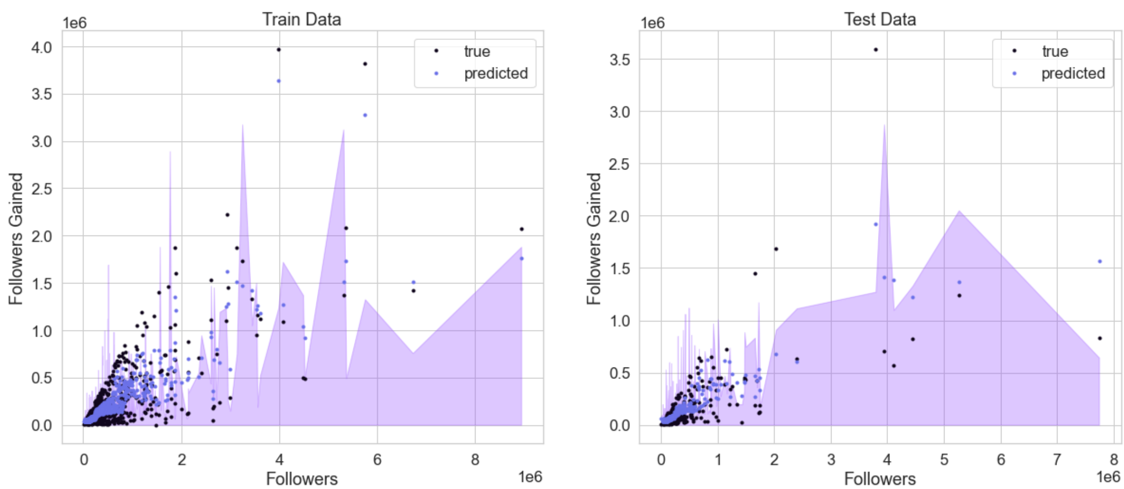
[71]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 8))

# Customize which model is actually being plotted
ax1.set_title("Poisson Model Performance on Test set")
ax1.scatter(y_test, pr.predict(X_test_ss),
                alpha=0.5, label="model output", color=ppt_colors[1])

ax2.set_title("K-neighbors Model Performance on Test set")
ax2.scatter(y_test, gs_knn.predict(X_test),
                alpha=0.5, label="model output", color=ppt_colors[2])

# Same setup for both plots (x and y labels, line showing y=x)
y_equals_x = np.linspace(0, 4e6)
for ax in (ax1, ax2):
    ax.set_xlabel("True")
    ax.set_ylabel("Predicted")
    ax.plot(y_equals_x, y_equals_x, label="predicted = actual", color="black")
    ax.legend()

plt.savefig("../../reports/figures/Models_performances_2.png",␣
 ↪bbox_inches='tight', dpi=400)
```



Final Model Performances

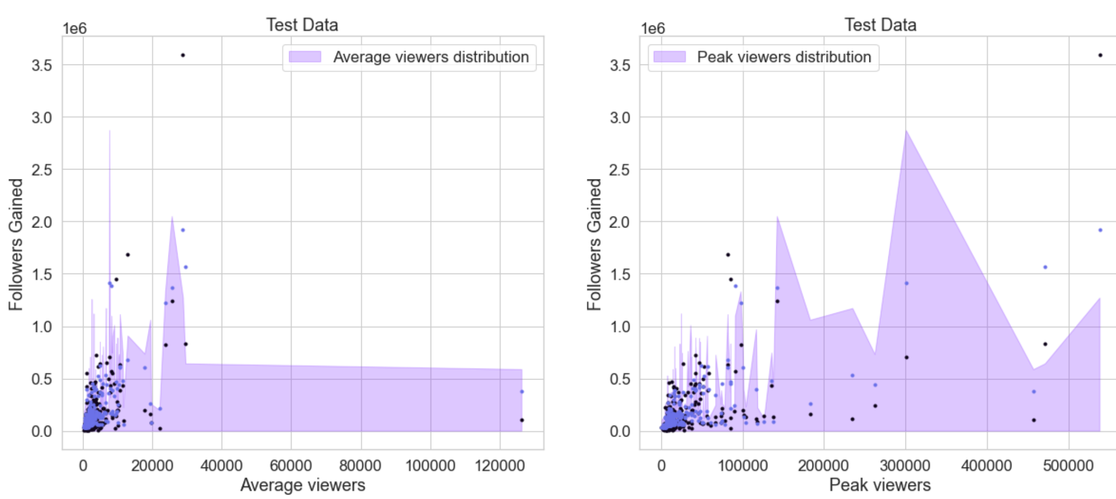
[67]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))
fig.subplots_adjust(hspace=0.3, wspace=0.2)

plot_obs_pred(
    df=df_train,
```

```
    feature="Followers",
    weight="Watch time(Minutes)",
    true="Followers gained",
    predicted=rfr_final.predict(X_train),
    y_label="Followers Gained",
    title="Train Data",
    ax=ax[0],

)

plot_obs_pred(
    df=df_test,
    feature="Followers",
    weight="Watch time(Minutes)",
    true="Followers gained",
    predicted=rfr_final.predict(X_test),
    y_label="Followers Gained",
    title="Test Data",
    ax=ax[1],

)

plt.savefig("../../reports/figures/Random_Forest_performances_1.png",␣
 ↪bbox_inches='tight', dpi=400)
plt.show()
```



[68]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))
fig.subplots_adjust(hspace=0.3, wspace=0.2)

plot_obs_pred(
```

```
    df=df_test,
    feature="Average viewers",
    weight="Watch time(Minutes)",
    true="Followers gained",
    predicted=rfr_final.predict(X_test),
    y_label="Followers Gained",
    title="Test Data",
    ax=ax[0],
    fill_legend=True
)

plot_obs_pred(
    df=df_test,
    feature="Peak viewers",
    weight="Watch time(Minutes)",
    true="Followers gained",
    predicted=rfr_final.predict(X_test),
    y_label="Followers Gained",
    title="Test Data",
    ax=ax[1],
    fill_legend=True
)

plt.savefig("../../reports/figures/Random_Forest_performances_2.png", bbox_inches='tight', dpi=400)
plt.show()
```



3.0.5 Summary

Random Forest performed the best out of all my models with coefficient of determination: 60%. 60% of the test data fit the regression model.

```
[ ]:
```